

版权相关注意事项：

1、书籍版权归著者和出版社所有

2、本PDF来自于各个广泛的信息平台，经过整理而成

3、本PDF仅用于寻找业务商业用途或者个人交流研究学习使用

4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负。

5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不适人目，质量低下，你也可以选择联盟退换整本PDF

6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得这边自喜，要经常翻阅！！经常翻阅

7、请于下载PDF后24小时内研究使用并删掉本PDF



版权相关注意事项：

1、书籍版权归著者和出版社所有

2、本PDF来自于各个广泛的信息平台，经过整理而成

3、本PDF仅用于寻找业务商业用途或者个人交流研究学习使用

4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负。

5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不适人目，质量低下，你也可以选择联盟退换整本PDF

6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得这边自喜，要经常翻阅！！经常翻阅

7、请于下载PDF后24小时内研究使用并删掉本PDF




版权相关注意事项：

1、书籍版权归著者和出版社所有

2、本PDF来自于各个广泛的信息平台，经过整理而成

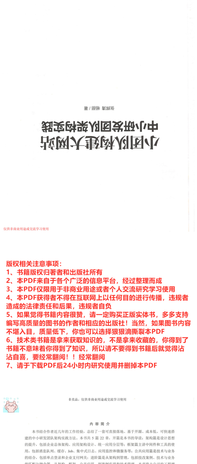
3、本PDF仅用于寻找业务商业用途或者个人交流研究学习使用

4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负。

5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不适人目，质量低下，你也可以选择联盟退换整本PDF

6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得这边自喜，要经常翻阅！！经常翻阅

7、请于下载PDF后24小时内研究使用并删掉本PDF



版权相关注意事项：

1、书籍版权归著者和出版社所有

2、本PDF来自于各个广泛的信息平台，经过整理而成

3、本PDF仅用于寻找业务商业用途或者个人交流研究学习使用

4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负。

5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不适人目，质量低下，你也可以选择联盟退换整本PDF

6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得这边自喜，要经常翻阅！！经常翻阅

7、请于下载PDF后24小时内研究使用并删掉本PDF




版权相关注意事项：

1、书籍版权归著者和出版社所有

2、本PDF来自于各个广泛的信息平台，经过整理而成

3、本PDF仅用于寻找业务商业用途或者个人交流研究学习使用

4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负。

5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不适人目，质量低下，你也可以选择联盟退换整本PDF

6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得这边自喜，要经常翻阅！！经常翻阅

7、请于下载PDF后24小时内研究使用并删掉本PDF




版权相关注意事项：

1、书籍版权归著者和出版社所有

2、本PDF来自于各个广泛的信息平台，经过整理而成

3、本PDF仅用于寻找业务商业用途或者个人交流研究学习使用

4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负。

5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不适人目，质量低下，你也可以选择联盟退换整本PDF

6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得这边自喜，要经常翻阅！！经常翻阅

7、请于下载PDF后24小时内研究使用并删掉本PDF

非卖品，仅供非商业用途或交流学习使用

# 目录





仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用







仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用







仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用







仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用







仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用







仅供非商业用途或交流学习使用

仅供非商业用途或交流学习使用

# 第1篇

# 开　篇

非卖品，仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 1 可参考的才是有价值的（含案例和代码）

技术大会上的分享大多“高大上”——亿级流量、超大型研发团队，虽然值得借鉴，但由于应用场景与研发资源的差异，一般企业并不容易落地。其实，中小型研发团队在 IT 行业还是占大多数，他们在技术架构方面的问题较多，技术阻碍业务、跟不上业务发展的情况很常见。笔者是一个有十多年经验的 IT 老兵，曾在两家几千人的技术团队做过架构与技术管理工作，也曾在几十人至几百人的中小研发团队做过首席架构师和 CTO。在互联网大厂做技术研发工作，大多数人只是一个“螺丝钉”。而在中小研发团队做研发工作，则比较容易掌控全局。本书结合笔者近几年的工作经验，摸索出一套可直接落地、基于开源、成本低、可快速搭建的框架及架构方案。本书贴近一般程序员的实际情况，更具应用和参考的价值，共 5 篇 22 章，包括开篇、框架篇、架构篇、公共应用篇和进阶篇，以下是具体介绍。

## 1.1 框架篇——工欲善其事，必先利其器

如果说运维是地基，那么框架就是承重墙。盖房子是先打地基，再建承重墙，最后才垒砖，所以中间件的搭建和引进是建设高可用、高性能、易扩展、可伸缩的大中型系统的前提。框架篇中的每一章主要由四部分组成：基本概念、工作原理、使用场景及可

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



直接调试的 Demo。其中中间件及 Demo 历经两家公司四年时间的考验，涉及几百个应用、100 多个库和 1 万多张表，日订单从几万到十几万，年 GMV 从几十亿到几百亿。所有中间件与工具都基于开源。早期也有部分中间件和工具是自主研发的，比如集中式日志和度量框架，后期在第二家公司时为了快速搭建、降低成本、易于维护和扩展，全部改为开源软件。这样不仅利于个人的学习成长、知识重用和职业生涯发展，也利于团队的组建和人才的引进。

### 1. 集中式缓存 Redis

缓存是计算机的难题之一，分布式缓存亦是如此。Redis 看起来非常简单，但它影响系统的效率、性能和数据一致性。用好它不容易，包括缓存时长（复杂多维度的计算）、缓存失效处理（主动更新）、缓存键（Hash 和方便人工干预）、缓存内容及数据结构的选择、缓存雪崩的处理、缓存穿透的处理等。Redis 除了缓存的功能，还有其他功能，比如 Lua 计算能力、Limit 与 Session 时间窗口、分布式锁等。我们使用 ServiceStack.Redis 做客户端，使用方法详见 Demo。

### 2. 消息队列 RabbitMQ

消息队列好比葛洲坝，有大量数据的堆积能力，再可靠地进行异步输出。它是 EDA 事件驱动架构的核心，也是 CQRS 同步数据的关键。为什么选择 RabbitMQ 而没有选择 Kafka？是因为业务系统对消息有高可靠性要求，以及对复杂功能（如消息确认）的要求。

### 3. 集中式日志 ELK

日志主要分为系统日志和应用日志两类。试想一下，如何在一个具有几百台服务器的集群中定位问题？如何追踪每天产生的几 GB 甚至几 TB 的数据？集中式日志就是此类问题的解决方案。早期我们使用自主研发的 Log4Net+MongoDB 来收集和检索日志信息，但随着数据量的增加，查询速度却变得越来越慢。后期改为开源的 ELK，虽然易用性有所下降，但它支持海量数据及与编程语言无关的特征。

### 4. 任务调度 Job

任务调度 Job 如同数据库作业或 Windows 计划任务，是分布式系统中异步和批处理的关键。我们的 Job 分为 WinJob 和 HttpJob：WinJob 是操作系统级别的定时任务，使用开源的框架 Quartz.NET 实现；而 HttpJob 则是自主研发实现的，采用 URL 方式可定时调



仅供非商业用途或交流学习使用

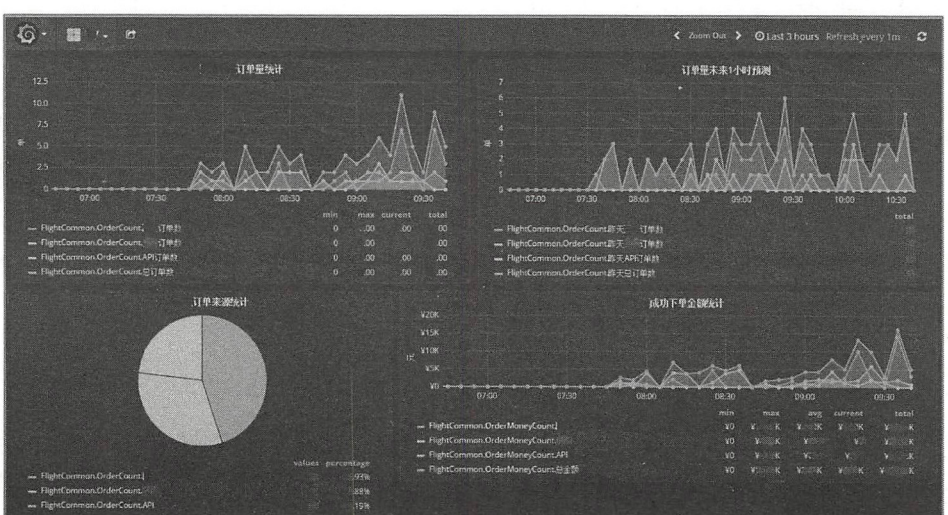
非卖品，仅供非商业用途或交流学习使用



用微服务。HttpJob 借助集群巧妙地解决了 WinJob 的单点和发布问题，并集中管理所有的调度规则，调度规则有简单规则和 Cron 表达式。HttpJob 简单易用，但间隔时间不能低于 1 分钟，毕竟通过 URL 方式来调度并不高效。下图是 HttpJob 的管理后台。


## 5. 应用监控 Metrics

“没有度量就没有提升”，度量是改进优化的基础，是做好一个系统的前置条件。Zabbix 一般用于系统级别的监控，Metrics 则用于业务应用级别的监控。业务应用是个“黑盒子”，通过数据埋点来收集应用的实时状态，然后展示在大屏或看板上。它是报警系统和数字化管理的基础，还可以结合集中式日志来快速定位和查找问题。我们的业务监控系统使用 Metrics.NET+InfluxDB+Grafana，如下图所示。





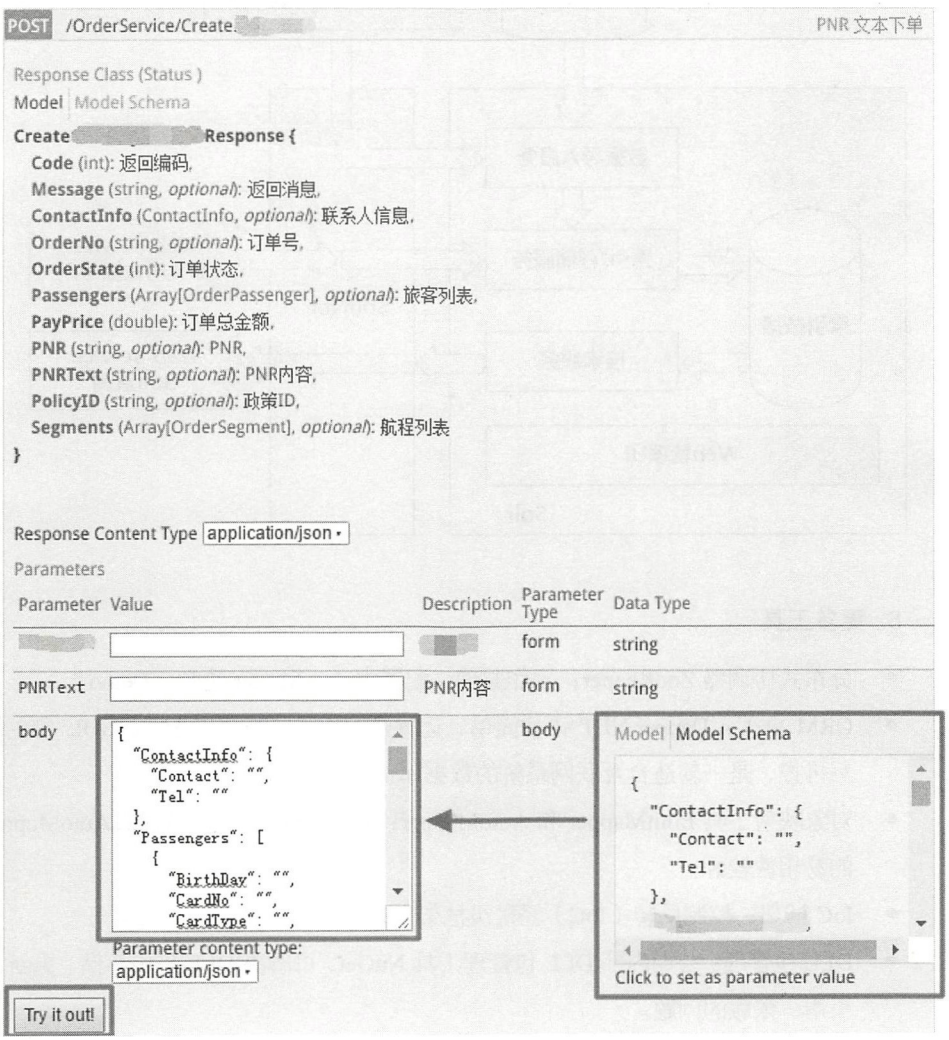

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 6. 微服务框架 MSA

微服务是细粒度业务行为的重用，需要与业务能力及业务阶段相匹配。微服务框架是实现微服务及分布式架构的关键组件，我们的微服务框架是基于开源 ServiceStack 实现的。它简单易用、性能好，文档自动生成、方便调试测试，调试工具是 Swagger UI，自动化接口测试工具是 SoapUI。微服务的接口开放采用自主研发的微服务网关，配置治理后台即可。网关以 NIO、IOCP 的方式实现高并发，主要功能有鉴权、超时、限流、熔断、监控等。下图是 Swagger UI 调试工具。





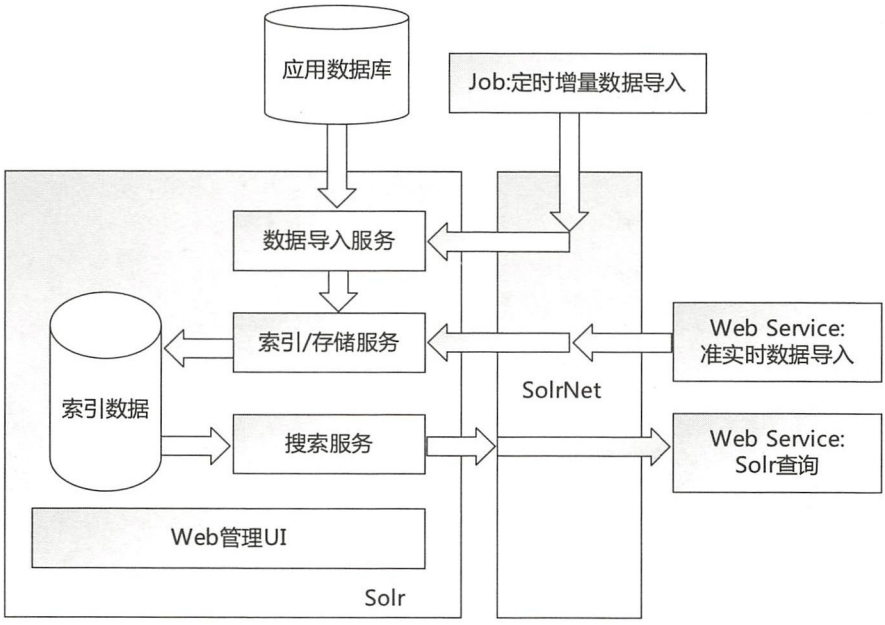

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 7. 搜索服务 Solr

分库分表后的关联查询，大段文本的模糊查询，这些要如何实现呢？显然传统的数据库没有很好的解决办法，这时可以借助专业的检索工具。全文检索工具 Solr 不仅简单、易用、性能好，而且支持海量数据高并发，只需实现系统两边数据的准实时或定时同步即可。下图是 Solr 的工作原理。




### 8. 更多工具

- 分布式协调器 ZooKeeper：工作原理、配置中心、Master 选举、Demo。
- ORM 框架：Dapper.NET 语法简单、运行速度快，与数据库无关，SQL 自主编写可控，是一款适合互联网系统的数据库访问工具。
- 对象映射工具 EmitMapper 和 AutoMapper：EmitMapper 的性能较高，AutoMapper 的易用性较好。
- IoC 框架：控制反转（IoC）轻量级框架 Autofac。
- DLL 包管理：公司内部 DLL 包管理工具 NuGet，可解决 DLL 集中存储、更新、引用、依赖的问题。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



- 发布工具 Jenkins：一键编译、发布、自动化测试、一键回滚，高效、便捷、故障率低。

## 1.2 架构篇——思想提升

会使用以上框架并不一定能成为优秀的架构师，但优秀的架构师一定会使用框架。架构师除了会使用工具，还需要有架构设计思想和性能调优技能。架构篇以真实项目为背景，设计方法追求简单有效，内容包括企业总体架构、应用架构设计、统一应用分层、调试工具 WinDbg。

### 1. 企业总体架构

当我们有了几百上千个应用后，不仅需要单个项目的架构设计，还需要企业总体架构做顶层思考和指导。大公司的技术人员比较难看到商业全貌，而小公司又缺乏客户流量和中间件的应用场景，中型公司则兼而有之，企业总体架构相对来说容易落地。企业总体架构需要在技术、业务、管理之间游刃有余地切换，包括业务架构、应用架构、数据架构和技术架构。1.5 节是一个脱敏感信息后的真实案例，参考了 TOGAF 标准，但内容以解决公司系统的架构问题为导向、以时间为主线，包括企业商务模型、架构现状、架构规划和架构实施。

### 2. 应用架构设计

应用架构设计如同施工图纸，能直接指导工程代码的实施。上一环是功能需求，下一环是代码实施，这是架构设计的价值所在。从功能需求到用例、用例活动图、领域图、架构分层、核心代码，它们之间环环相扣。做不好领域图可能源自没有做好用例活动图，因为用例活动图是领域图的上一环。关注职责、边界、应用关系、存储、部署是架构设计的核心。

### 3. 统一应用分层

给应用分层这件事情很简单，但是让一家公司的几百个应用采用统一的分层结构，这可不是一件简单的事情。它要做到可大可小、简单易用、支持多种场景。我们使用 IPO 方式实现：I 表示 Input、O 表示 Output、P 表示 Process，一进一出一处理。应用系统的本质就是机器，是处理设备，也是一进一出一处理，IPO 方式相对于 DDD 而言更简单实用。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 4. 诊断工具 WinDbg

生产环境偶尔会出现一些异常问题，而 WinDbg 或 GDB 就是解决此类问题的利器。调试工具 WinDbg 如同医生的听诊器，是系统“生病”时进行问题诊断的逆向分析工具。Dump 文件类似于飞机的黑匣子，记录生产环境程序运行的状态。诊断工具一章主要介绍 WinDbg 和 ProcDump 的使用，并分享一个真实的案例。多年前不知谁写的代码，导致每一两个月偶尔出现 CPU 飙高的现象。我们先使用 ProcDump 在生产环境中抓取异常进程的 Dump 文件，然后在不了解代码的情况下通过 WinDbg 命令进行分析，最终定位到有问题的那行代码，如下图所示。


## 1.3 公共应用篇——业务与技术的结合

先工具再框架，然后架构设计，最后深入公共应用。公共应用因为与业务系统结合紧密，但又具有一定的独立性，所以一般自主开发，不使用开源软件也不方便开源。公共应用主要包括单点登录（SSO）、企业支付网关、CTI 通信网关（短信、邮件、微信）等，下面介绍单点登录和企业支付网关。

### 1. 单点登录

应用拆分后总要合在一起，拆分是应用实施层面的拆分，合成是用户层面的合成，而合成必须解决认证和导航问题。单点登录即只需要登录一次，便可到处访问，它建立在用户系统、权限系统、认证系统和企业门户的基础上。我们的凭证数据 Token 使用 JWT 标准，以解决不同语言、不同客户端、跨 Web API 的安全问题。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 2. 企业支付网关

企业支付网关集中和封装了公司的各大支付系统。例如，支付宝、财付通、微信、预付款等。它统一了业务系统调用各支付接口的方式，简化了业务系统与支付系统的交互。它将各种支付接口统一为支付、代扣、分润、退款、退分润、补差、转账、冻结、解冻、预付款等，调用时只需选择支付类型即可。企业支付网关将各大支付系统进行集中地设计、研发、部署、监控、维护，提供统一的加解密、序列化、日志记录和安全隔离服务。

## 1.4 进阶篇——从架构到管理

架构要落地、固化和提升，需要通过技术架构与组织架构的对齐来实现。从生产力到生产关系，从架构师到技术管理，我们关注的角度也将发生变化。从关注技术到关注技术的商业价值，技术与业务的匹配与融合，技术团队的文化，等等。本篇内容包括技改之路、技术与业务的匹配与融合、研发团队文化是怎么“长”出来的等 5 章，下面重点介绍其中 3 章。

### 1. 技改之路：从单体应用到微服务

技改是技术改造的简称，是技术的蜕变。第 18 章介绍在公司技术发展的某个瓶颈阶段，按原有开发和组织方式已经无法“玩下去”，这时公司希望引进架构师或技术牛人来破解当前困局。技改之路少讲技术多讲路，我们不过多地关注技术细节和中间件的实现，而重点讲述技术改造的过程和思考。技改是大折腾，于公司于个人而言都是。“小改怡情、大改伤身”，所以真正的高手下棋，应该是通盘无妙招，让正确的事情很容易发生，基于自然的演化来实现技术的演进。

### 2. 技术与业务的匹配与融合

是什么在驱动公司的发展？技术人员说“科学技术是第一生产力”，市场人员说“没有市场，哪来的业务”，运营人员说是他自己。应该说他们都是正确的，但又不全面。这如同盲人摸象一样，引发了不少的争论，也直接或间接地导致了技术人员与业务人员的矛盾。第 21 章先抛出了一个启发性的问题，然后分析问题出在哪里，理解源于彼此的了解，如何去匹配与融合，最后正面回答了该问题。只有尊重事物的发展规律，加强技



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



术与业务之间的合作，才能促进公司的发展。

### 3. 研发团队文化是怎么“长”出来的

从死气沉沉到激情活力，从故步自封到好学分享，这是一个有关团队文化的主题。寺庙文化传承千百年，舌尖上的美食流传至今，它们是如何形成和生长的？是参考大公司或从管理书籍上挑选几个词语，还是脚踏实地，自己一步一步埋头干？第 22 章先介绍我们曾遇到的问题，然后是解决之道，包括管理工具、制度和行为措施，并予以贯彻，形成一种习惯，最后总结并归纳成几个可以贴到墙上的大字，即“共治分享自视一起拼，简单有效快”，这个过程就如同花朵一般。只有这样“长”出来的文化，才能管人做事，成为公司或团队的核心竞争力。

以上顺序不仅是架构改造的参考路径，也是架构师的成长路径。照着做，你也能成为架构师。但为了本书的阅读效果，篇章目录并没有采用以上顺序，而是先介绍架构，然后是框架、公共应用和进阶部分。

根据我们以往的经验，分享者主讲一个小时左右，业务研发人员就可以快速地进入项目实战。后面新加入的团队成员也可通过 Wiki 自主快速学习。这是我们之前对自己的要求，尽量降低工具对研发人员的门槛，简单实用、降低成本。本书中部分 Demo 采用 C#、Java 和 Go 语言，但到了框架与架构层面，与语言本身没有太多关系，如 RabbitMQ、Job、Redis 和集中式日志 ELK，服务端的部署都是一样的，只是客户端语言版本稍有不同。所有 Demo 在一段时间内都可直接运行，服务地址和管理后台也可直接访问。以上这些基础工作，希望能够帮助中小型研发团队，解决读者在项目中遇到的实际问题，也愿与读者一起在架构方面有所成长！

## 1.5 案例参考和 Demo 下载

- 所有案例和 Demo 的下载地址：https://github.com/das2017?tab=repositories



仅供非商业用途或交流学习使用

仅供非商业用途或交流学习使用

# 第 2 篇

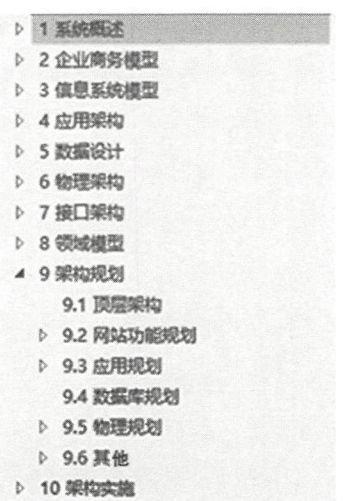
# 架构篇

非卖品，仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 2 企业总体架构

企业总体架构是什么，有什么用，具体怎么做呢？以笔者曾任职的公司为案例，一起来探讨这个问题。这家公司当时有 200 多个研发人员和 200 多台服务器，笔者刚进这家公司时，他们的系统总是出现各种问题。例如，日常发布系统时或访问量稍微过大时，系统就会出现很多故障，而且找不到故障发生的根本原因。笔者进公司后的主要任务就是对这个系统进行升级改造，花了一个半月的时间写了一份企业总体架构设计文档，文档总共有 124 页，指导了之后的技术改造，下图是那份文档的目录。




仅供非商业用途或交流学习使用

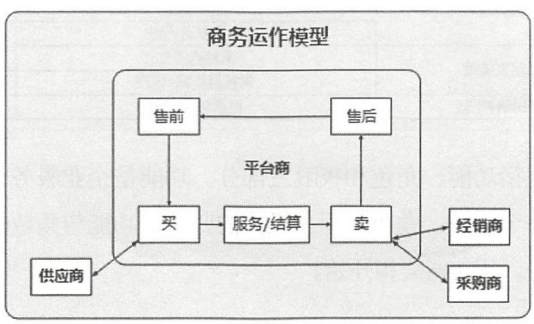
非卖品，仅供非商业用途或交流学习使用



## 2.1 企业商务模型

企业商务模型的主要内容包括主营业务、商务模式、商务主体、竞品分析、组织架构、商务运作模型和业务流程等。

主营业务即公司做什么业务，商务模式即公司怎么赚钱，商务主体即哪几个人在一起做这门生意，竞品分析即了解竞争对手的情况，组织架构即公司部门是怎么划分的。在组织架构图中标出人数，根据系统与业务之间的对应关系，可以了解系统中哪些模块使用的频率高，以及业务与其对应模块的复杂度。商务运作模型即公司是如何运作的，售前做计划，找供应商把东西买进来后，经过服务和结算，再卖给经销商和采购商，使我们获得利润，售后进行大数据分析后又指导我们的售前，整个过程形成良性循环。可以把一家公司想象成一台机器，输入的是钱，转一转后，又能够生出更多的钱出来。商务运作模型如下图所示。




最后是业务流程和附档资料。业务流程包括预订流程、订单处理流程、产品供应流程、财务结算流程、账户管理流程。企业商务模型的建立，指导整个应用系统模型的建立，它是整个应用系统建设的基础和前提，毕竟应用系统是为业务服务的。

## 2.2 架构现状

架构现状的内容主要包括功能架构、应用架构、数据设计和物理架构。

### 1. 功能架构

采购商的功能如下图所示。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



**3.1 国内全部功能**

采购商的功能：

| 模块 | 功能 | 备注 |
| --- | --- | --- |
| 系统管理 | 资料信息 | |
| | 投诉，建议 | |
| 机场服务 | 代换登机牌供应管理 | |
| | 已确认订单代换处理 | |
| | 换登机牌管理 | |
| | 代换登机牌 | |
| 采购机票管理 | PNR 导入创建订单 | |
| | 航班查询及预订 | |
| | 团队票申请 | |
| | 申请改签及升舱查询 | |
| | 三字代码查询 | |
| 在线订单管理 | 当日最新订单 | |
| | 所有订单查询 | |
| | 采购报表下载 | |
| | 自由转账 | |
| | 退废票相关查询 | |
| | 申请行程单及查询 | |
| | 行程单领用及管理 | |
| | 保险管理 | |
| 短信平台 | 短信充值 | |
| | 短信发送 | |
| | 短信发送历史 | |
| 常旅客管理 | 常旅客添加 | |
| | 常旅客查询/修改 | |
| 常用软件下载 | 机票软件下载 | |

功能架构主要包括功能、角色和权限三部分。功能是企业服务，用户使用的每一个功能就是企业的每一个服务。角色是用户操作的归类，功能与角色的对应关系即权限。了解系统架构的现状，从功能架构开始。

### 2. 应用架构

应用就是处理器，应用架构的内容包括现有架构图、Web 应用现状、作业小应用（Job）现状和接口架构。其中，接口是应用层面的关键，它是一个程序与另外一个程序交互的部分。

应用架构图（详见位于 18.1 节第 2 点的那张图）列出了哪些业务逻辑没有被重用，换句话说，业务逻辑被多少个应用调用，就需要被重复开发多少次，一旦改了一个地方，就要同时改多个地方，导致系统开发效率非常低下。各业务逻辑如预订逻辑，虽然被多个应用调用，但它与应用是没有关系的，业务逻辑可以独立存在，也可以寄宿于多个应用。业务逻辑是一个业务操作的抽象，而业务应用与业务部门共同完成了业务操作。



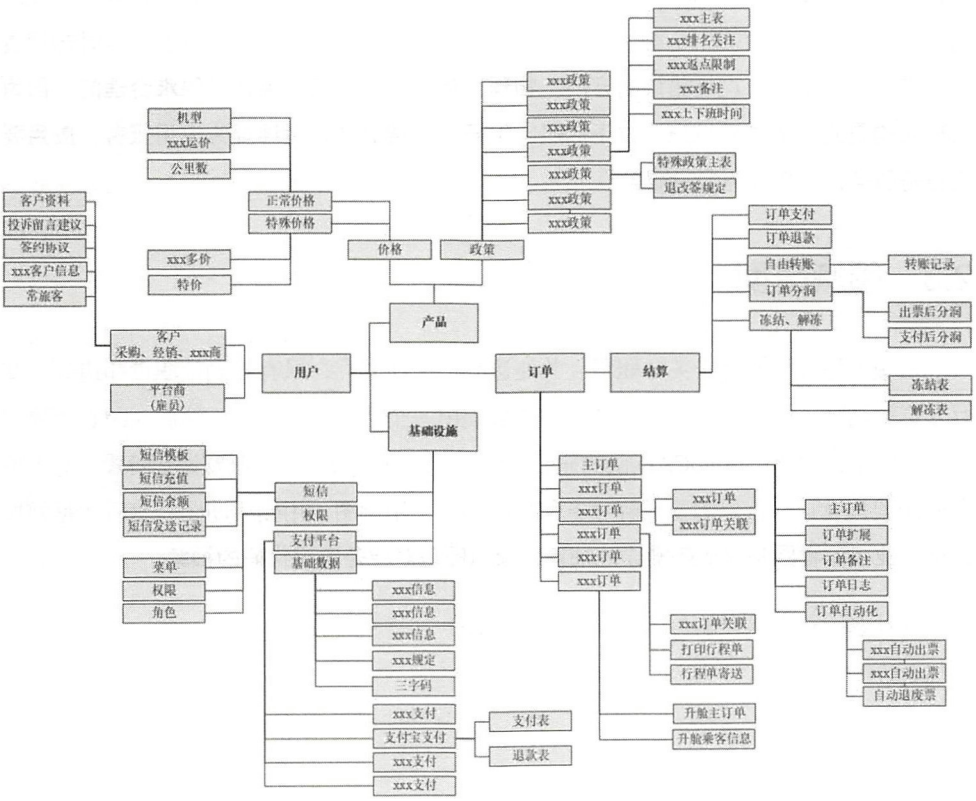
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 3. 数据设计

100 多个数据库，一万多张表，能否使用一张 E-R 图来表示呢？可以的。数据设计依赖于企业的数据，而不是数据库的设计，将企业数据适当归类，会直接导致数据设计，最终画出 E-R 图，数据设计完成后，数据库设计就自然而然出来了。超越库、超越表去看这张 E-R 图，可以看出它包括产品、订单、结算、用户、基础设施这五类数据。低层的 E-R 图可以变，但是高层的 E-R 图一般不会变化，因为它是根据业务模型而定的，业务模型稳定，高层 E-R 图也是稳定的。数据库只要早期设计得好，是可以做到易伸缩、易拆分的。下图从内往外看，一个框既可以是一个库，也可以是一个模块，还可以是一个表。在业务发展的早期它可以是一个库，里面有 5 个模块，中期可以分为 5 个库，后期以更低级别可以分为更多的库，这与业务阶段及系统复杂度相关。在数据的设计完成后，数据库的设计也就很容易进行规划和调整。




以上是数据库、数据表之间的静态关系，接下来我们介绍数据的流转状态，即状态



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



图。通过数据状态图去了解现有数据的流转变迁，如位于 18.3 节【数据变迁】部分的那张国内订单状态变迁图，这种图的价值不仅在于数据库层，还在于服务化。从等待支付到支付成功，中间有一个支付行为，通过这个支付行为把数据状态变更为支付成功，否则继续等待，直到超时关闭订单。这个支付行为可以做成一个微服务，然后由不同的应用去调用。

#### 4. 物理架构

物理架构的内容主要包括 IDC 机房、机房之间的访问关系、机房内服务器物理部署图、机房与业务分布、网站架构、数据库架构、集群清单和域名清单。将这些内容以列表和图形方式整理出来，就很容易了解和发现问题，只有发现问题才能解决问题，特别是在全局体系架构方面，这也是表和图的价值所在。当时这家公司共有 5 个地区、8 个机房，虽然只有 200 多台服务器，但分布很散，导致物理结构复杂，通信也很复杂。技改前故障不断，主要的原因就是物理架构不合理，运维要占 60%～70%的责任，当时却把责任归咎为应用架构，这是错误的方向。**物理架构不合理，应用架构是很难合理的，因为物理架构是我们的基础设施，位于底层，下层为上层服务，运维要为应用服务，应用要为业务服务，业务要为客户服务。**

## 2.3 领域模型

领域模型关注概念、关注职责、关注边界、关注交互，只有先确定职责和边界，交互才会很清晰。领域模型是针对现有问题域提出一个系统解决方案，然后在图表上建立完整的模型，如同用 AutoCAD 画的施工图纸一样。领域模型属于概要设计阶段，对于单个应用架构设计，首先需要了解业务和功能需求、用例图、用例活动图，然后才是领域模型。**业务流程图是对业务操作的抽象，领域图是对业务逻辑代码的抽象。**

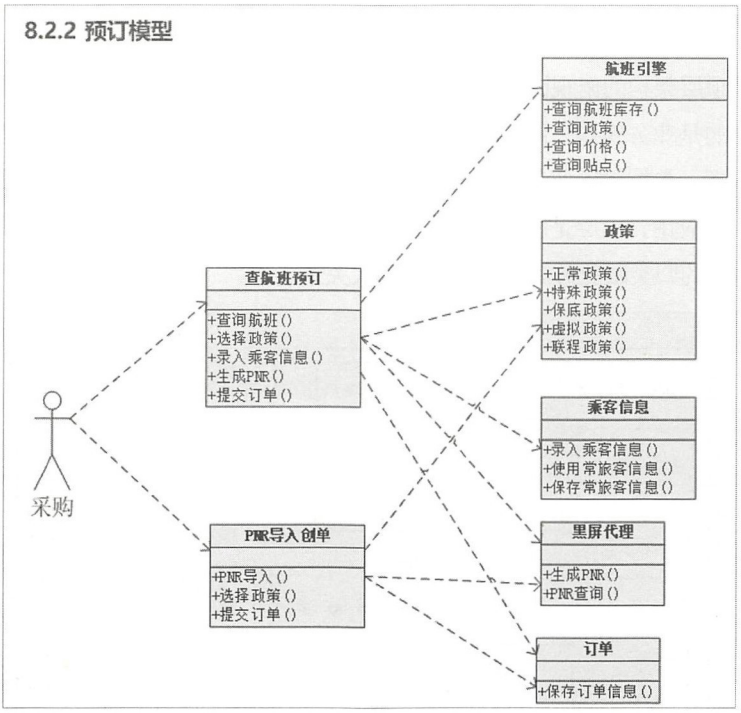
预订模型如下图所示。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






建立领域词汇是建立领域模型的第一步，它能统一词汇、明确概念，以减少一词多义、一义多词的情况。概念一旦确定，再扩展属性和行为，然后把它当作一个单元与其他事物构建在一起，就很容易形成模型，领域模型与企业商务模型中的业务流程图有对应关系。**领域模型在实现时可大可小，在业务的早期，系统比较小的情况下，它有可能是一个类。当系统做大了以后，它可能是一个 DLL 库。再做得更大一点的时候，它可能是一个服务，给不同的应用去调用。每一个方法都有成为服务的潜质，特别是在系统的中后期。**领域模型是业务逻辑代码的施工图纸，它不仅有利于我们对现有系统业务逻辑的了解，也指导未来的架构改造。

## 2.4 架构规划

当我们了解业务和架构的现状，发现现有架构的问题之后，接下来就可以做中远期架构规划，以及架构的调整和具体实施。架构规划内容包括顶层架构规划、网站功能规划、应用规划、SOA 规划、分层架构规划、数据库规划和物理规划等。



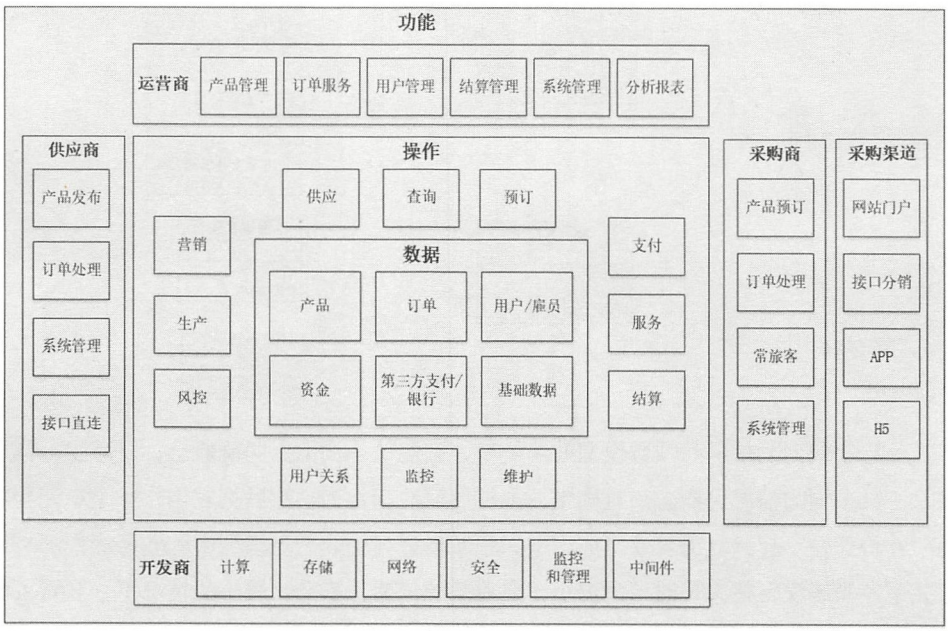
仅供非商业用途或交流学习使用

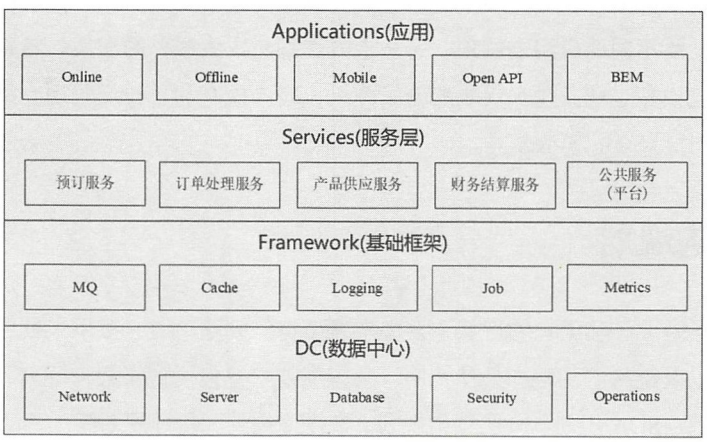
非卖品，仅供非商业用途或交流学习使用



### 1. 顶层架构规划

以下是顶层架构的俯视图和剖面图。**第一张图是俯视图**，整个顶层架构最外层的是功能，中间的是业务操作，内层的是数据。功能对应业务系统的用户界面，操作对应业务系统的服务，数据对应业务系统的数据存储，如数据库。**第二张图是剖面图**，切一刀来看，上层是应用，中层是服务和框架，下层是基础设施数据中心。从图中的服务层可以看出，服务的归类跟业务流程的归类有很大关系。






仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

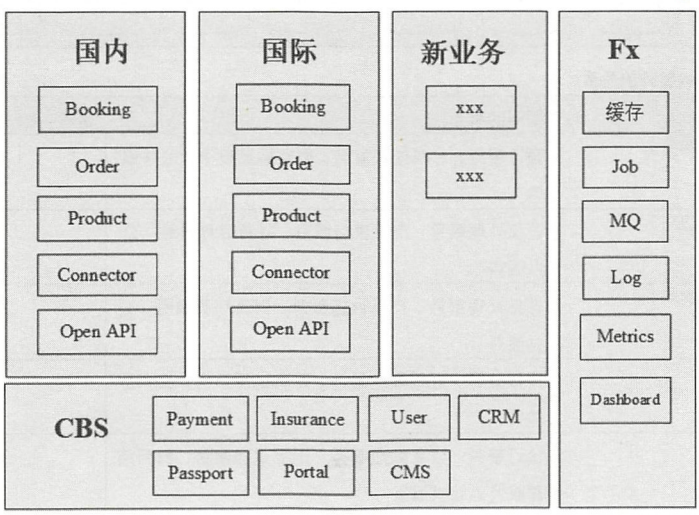


### 2. 网站功能规划

网站功能规划就是对功能的重新划分，对照架构现状，未来的功能应该如何调整？例如，案例中的国内网站功能规划，分别画出了全局功能图、采购商功能图、平台商功能图和供应商功能图。其实在做网站功能规划的时候，更多需要考虑现状，而不是未来调整的部分，如果没有很大问题，则不做调整，尊重历史。因为有些东西（如名称）用户已经使用很久了，调整往往比较难，合理大于准确。

### 3. 应用规划

**系统是什么？系统=元素+关系。应用架构是什么？应用架构=应用+架构。应用就是系统的最小单元，应用分类和应用编号构成了应用关系即应用的架构。**如下图中的案例，应用分类新建框架 Fx 和公共业务系统 CBS，原有的 200 多个应用中并没有这两个产品线，而是分布在了不同的业务线中，从而导致重复建设。应用编号是给每个应用分配一个六位的数字 ID，如同我们的身份证一样，头两位表示产品线，中间两位表示子系统，最后两位表示应用，如 100206。应用编号是应用管理、依赖和追踪的基础，集中式日志和监控框架都使用了应用编号。




### 4. SOA 规划

SOA 规划就是接口规划，它的归类与商务模型中的业务流程有对应关系。下图中的案例有五个服务中心：预订服务、订单处理服务、产品供应服务、财务结算服务和公共

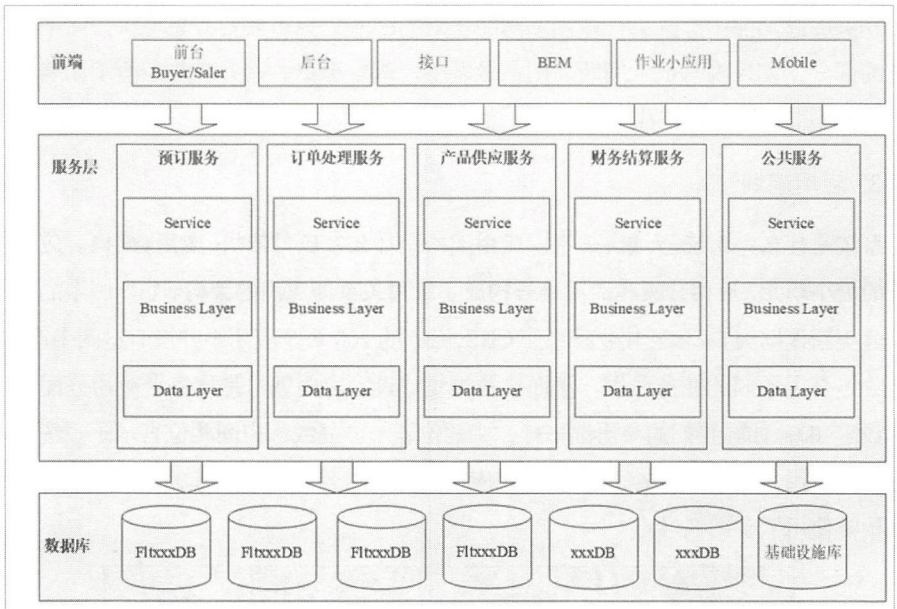


仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



服务。每个服务只需要实现一套自己的逻辑，前台、后台、接口、作业小应用等都可以调用，服务的逻辑跟业务逻辑是一致的，修改代码的时候只需要改一个地方就可以影响所有调用这服务的前端应用。




**前端应用与服务的关系：**

| 前端应用 | 调用的服务 | 备注 |
| --- | --- | --- |
| 前台（采购商） | 预订服务、订单处理服务、财务结算服务、公共服务 | |
| 前台（供应商） | 订单处理服务、产品供应服务、财务结算服务、公共服务 | |
| 后台（平台商） | 订单处理服务、产品供应服务、财务结算服务、公共服务 | |
| 接口 | 预订服务、订单处理服务、财务结算服务、公共服务 | |
| BEM | 预订服务、订单处理服务、产品供应服务、财务结算服务、公共服务 | |
| 作业小应用 | 订单处理服务、产品供应服务、财务结算服务、公共服务 | |
| Mobile | 订单处理服务、产品供应服务、财务结算服务、公共服务 | |



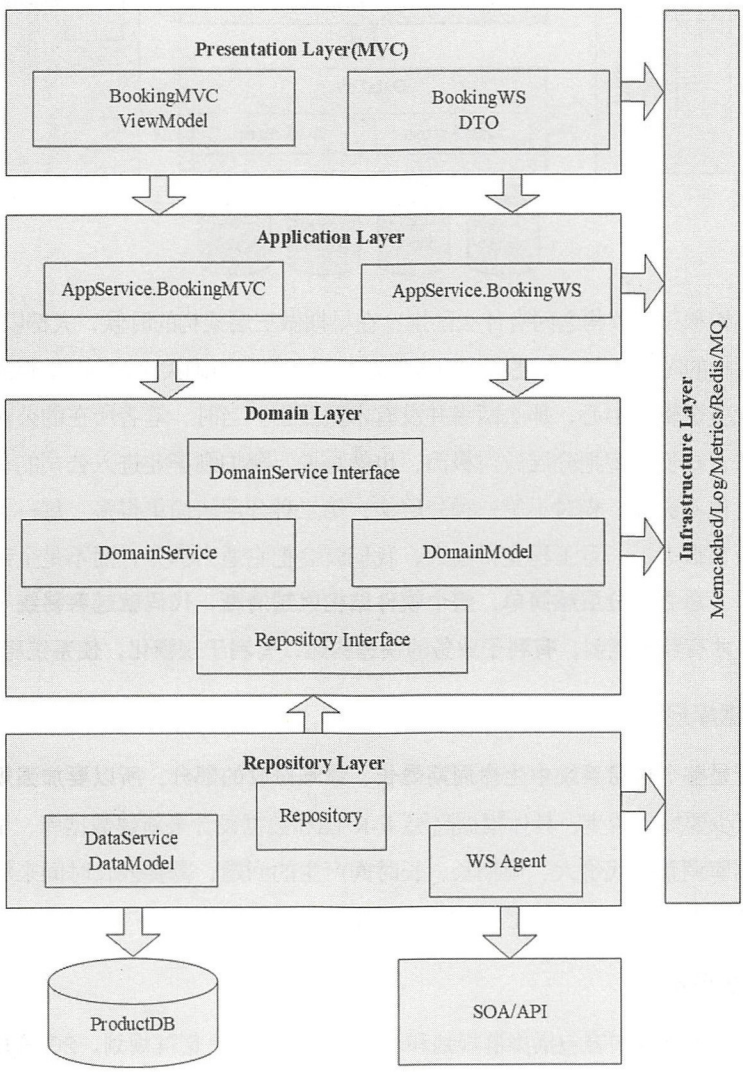
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 5. 分层架构

分层架构看似很简单，但保证整个研发中心都使用统一的分层架构就不容易了。那么如何保证整个研发中心都使用统一的分层架构，以达到提高编写代码效率、保证工程统一性的目的？先简单介绍当前两种比较流行的分层架构体系，一种是领域架构：包括仓储层（Repository Layer）、领域层（Domain Layer）、应用服务层（Application Layer）、表现层（Presentation Layer）和基础公共层（Infrastructure Layer），如下图所示。



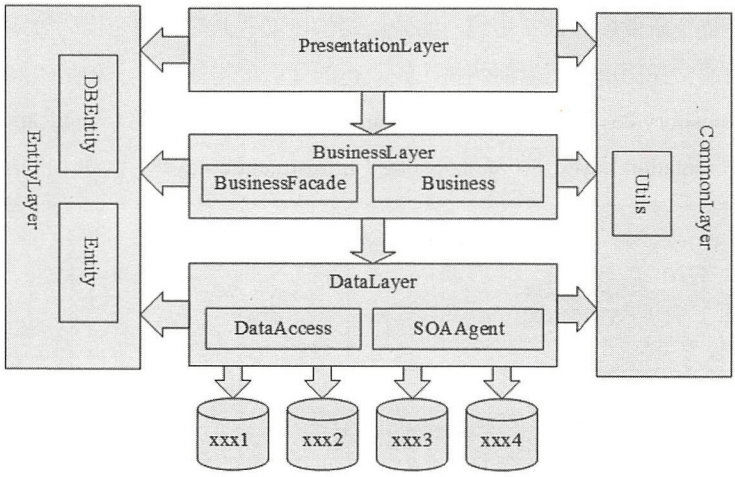



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



另一种是相对传统地分为三层：包括数据层（Data Layer）、业务逻辑层（Business Layer）和表现层（Presentation Layer），如下图所示。




领域架构和三层架构之间有什么区别？在早期做三层架构的时候，大都以表来驱动，在做领域架构的时候，大都以业务逻辑来驱动，两者的区别确实比较明显，但到了现在，如果都以业务逻辑为中心，那么两者并没有本质区别。当时，笔者所在的公司采用了第二种分层法，我们希望把分层做得极简，也就是说，哪怕刚毕业进入公司的员工，在分层时基本上也不会乱。相对于第一种分层法，第二种分层法简单得多。每一个应用的代码量都不应该很大，一旦工程变得过大，我们就会把它适当拆分，而不是全部放在一个单体应用里。**总之，分层越简单，整个软件结构就越清晰，代码就越容易统一。把工程做得极简，才有利于复制，有利于业务的快速构建，有利于规模化，使系统稳定可靠。**

### 6. 数据库规划

**数据库是整个信息系统中生命周期最长、最难修改的部分，所以要加强规划。**数据库的设计至少要提前两步，具体根据高层 E-R 图和数据设计来新建数据库，早建要比晚建好。数据库调整的代价大、周期长，长时间产生的问题，需要很长时间来解决。先在新库里解决新表，再根据当前业务和应用的需求，逐步调整旧表。

### 7. 物理规划

物理架构的规划内容包括集群规划和域名规划。首先是集群规划。**20 倍规划、5 倍**

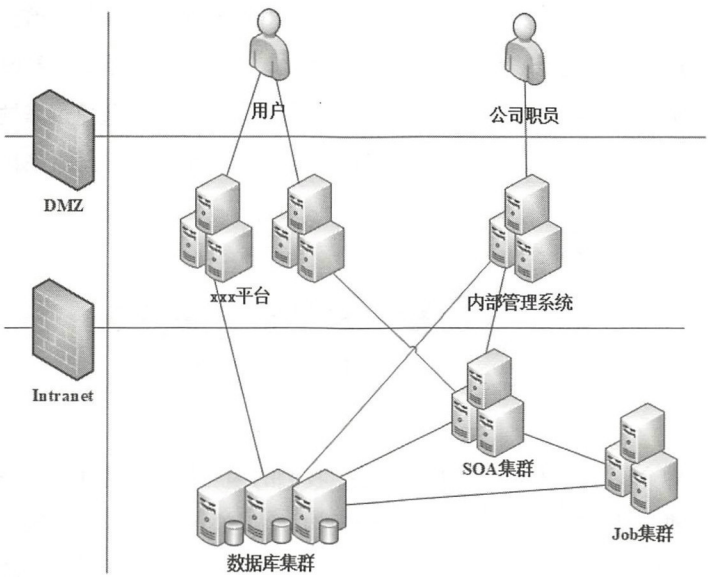


仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



**设计和 1.5 倍实施：**规划和设计要大一些，但实施时小一些，这样不仅便于将来的扩展，也节省了当前的费用。**两个逻辑网络：**一个内网和一个外网，两个负载均衡，两个防火墙，安全隔离内外网。**四条产品线：**国际、国内、新业务和公共业务，单点登录和企业支付网关等公共业务也属于一条产品线。**六个集群：**Web 集群、SOA 集群、中间件集群、数据库集群、Job 集群和 ITD 集群。以上横向集群与纵向产品线形成了一个矩阵结构，也基本确定了网络的基础架构。对于域名规划，对内的域名该改的改，该停用的停用，该合并的合并；对外的域名要尽量少改，要改也要有历史继承性（如跳转），要尽量减小对用户的影响。物理规划案例如下图所示。




### 8. 其他

除以上架构规划外，还有一些其他重要项，如源代码管理规划、文档管理规划、技术选型和团队分工。为什么还要做这些呢？因为统一了源代码怎么放、每个部门的文档怎么放、将来要用什么工具版本，才有利于团队的协作，基于统一的环境才能有更高层面的提升。对于团队分工，需要逐步对齐组织架构与系统的架构规划。对于技术选型，需要注意中间件的引进，要有节奏性，力量要相对集中，要小规模试点，找非核心项目，试用成功后再进行大规模推广。



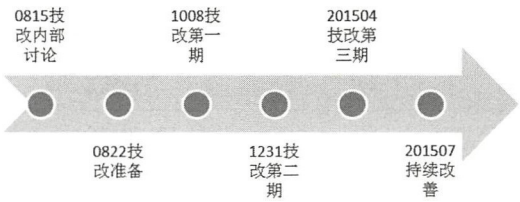
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



## 2.5 架构实施

做完架构规划后，就是架构实施落地了。我们的架构实施整体思路是：树目标、给地图、立榜样、抓重点、造文化、建制度、整环境、组建架构部。架构部内部招几名老程序员，外部招几个架构师。内部人员走出去，提高眼界。外部牛人请进来，落地了解历史和业务。技术建议是：SOA 服务化、基础设施平台化、公共业务服务化、加强项目概要设计。当研发团队达到 200 多人、有了几百个应用，且在故障不断的情况下，不能与以前一样没有设计就开始编码，而是要加强项目概要设计及评审。后面的补与前面的防，“两手都要抓，两手都要硬”。具体计划是：Roadmap 分步实施，改造一期、改造二期、改造三期，近细远粗、实事求是、逐步细化、逐步完善。不断立技改项目，不断将技改与业务研发项目相结合，技改即是工单、工单即是技改。避免过多地影响业务，并不断有业务价值输出，这是架构改造得以持续实施的关键！案例示意图如下。




以上简单地介绍了总体架构的编写方法，我们的编写思路是先了解业务，建立企业商务模型，主要包括静态的商务主体、组织架构、动态的商务运作模型和业务流程。再了解架构现状，建立现有信息系统模型，主要包括功能架构、应用架构、数据设计和物理架构。一个是商务，另一个是电子，两者是整个公司的电子商务系统。然后在企业商务模型和现有系统模型之上建立领域模型，领域模型相对稳定，直接指导接下来的架构规划，最后一定要落地，即架构实施。2.6 节是去掉敏感信息后的真实案例，它的价值如下：

- Big Picture（全局蓝图），起到方向性和指导性的作用。
- 将隐性知识显性化，方便传达、广而告之。
- 对于新员工的价值是快速入门。
- 对于老员工的价值是：了解全局，过程梳理，然后专注于自己的部分。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



关于企业总体架构，可以参考 TOGAF（开放组体系结构框架）。其实，我们在完成那份文档后才知道 TOGAF，它们之间有很多相似之处和不同之处。TOGAF 的内容主要包括业务架构、应用架构、数据架构和技术架构，而**我们当时只是以解决公司系统架构问题为导向、以时间为主线**，内容有企业商务模型、架构现状、领域模型、架构规划和架构实施。**方法论很重要，但看到事物本身的特点，深入问题及找到解决办法更重要。**

## 2.6 案例参考

- TopArch 案例参考地址：https://github.com/das2017/TopArchDemo



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 3 应用架构设计

有几个问题要与读者一起探讨。你做架构设计了吗？你认为要不要做架构设计？你的公司有没有做架构设计？在笔者得到的答案中，大部分人认为要做架构设计，但自己却很少做，自己经历的公司也很少做架构设计。这里是矛盾的，难道大部分人和公司都犯错了吗？应该不是这样。

## 3.1 初识架构设计

软件工程一般可分为需求、设计、编码、测试、部署和维护。既然架构设计是一个过程，那么就有输入和输出。架构设计输入的是 PRD（产品需求文档），输出的是架构设计文档，中间是处理过程和工具，具体如下。

- 输入：功能需求和非功能需求，从 PRD 中提取。
- 过程和工具：
  - 设计的目标和思路；
  - 功能设计——用例视图、用例活动图；
  - 应用——边界、逻辑架构、接口、领域图；
  - 数据存储；
  - 物理架构、安装部署；
  - 非功能设计。

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



- 输出：设计说明书，表述工具有 Word、Visio、UML 等。

需求是我要什么，即 What，而架构设计是我要怎么做，即 How。架构设计为施工阶段提供了指导，有利于接下来的编码、测试、部署和维护，包括项目排期、人员分工、配合、单元测试、物理部署、系统修改和升级。设计是施工的计划，没有计划就没有管理，计划可节约施工的成本和时间。如果没有架构设计就开始写代码，则会导致很多的问题，干着干着就干不下去了，或者干到一半必须得改等现象。

## 3.2 应用架构设计案例

以下是一个真实的应用架构设计案例，“国内航班查询引擎项目”的架构设计过程如下。

### 1. 功能清单

产品经理提供的 PRD 文档做得怎么样，第一眼就看它有没有功能清单。下图的功能清单主要有两个核心功能，一个是查询航班数据模块，另一个是清理缓存模块。

| 功能模块 | 主要功能 | 功能描述 |
| --- | --- | --- |
| 政策获取 | 航班查询（触发） | |
| | 航班查询 | 从黑屏及航班接口中查出符合条件航班 |
| | 取政策 | 从政策库取出政策 |
| | 取供应信息 | 出票标识，退票标识，出、退时间 |
| | 计算舱位价格 | 获取不同舱位的价格 |
| | 政策限制规则 | 地域限制、最高返点限制 |
| | 显示查询结果 | |
| 缓存策略管理 | 城市级别管理 | 城市级别，提供配置调整功能 |
| | 缓存管理 | 不同城市级别，缓存时间调整管理 |
| APP 采购贴点 | 贴点 | |

### 2. 用例图与用例活动图

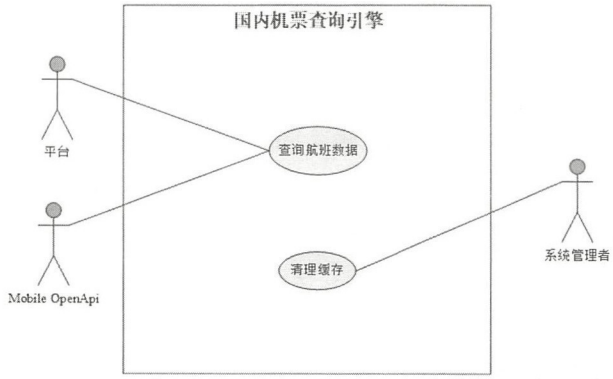
下图是用例图，用例图包含查询航班数据和清理缓存，这与功能清单有对应关系。每一个用例都可以展开为用例活动图，产品经理的活动图关注的是业务的逻辑，我们的用例活动图关注的是程序的业务逻辑，有更多的技术视角。



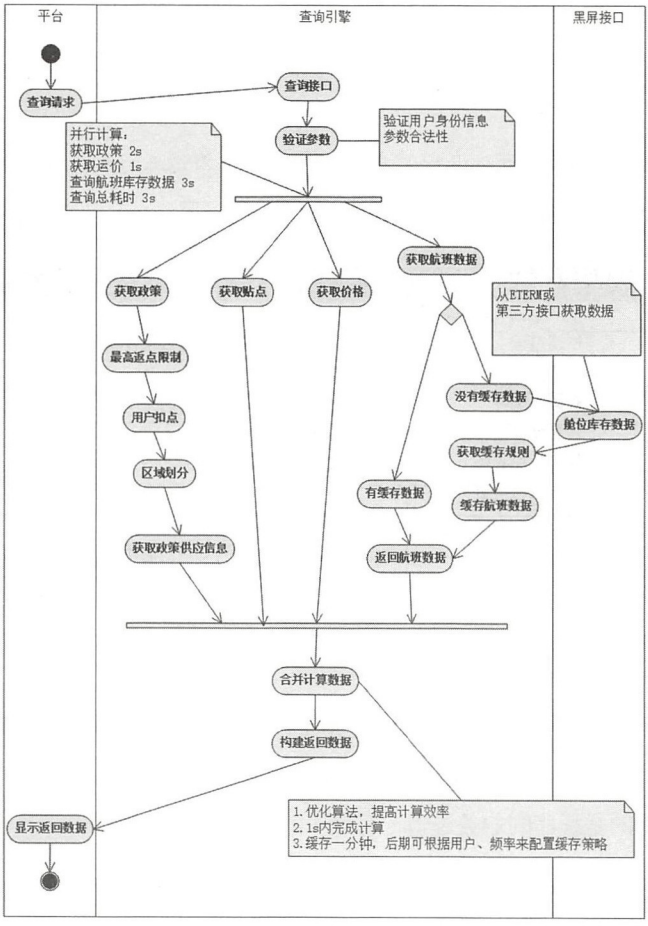
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






下图是用例活动图，前台网站或 Mobile 发起查询请求后，经过参数验证，然后分别获取政策、贴点、价格、航班数据，再合并计算数据，最后构建返回数据。





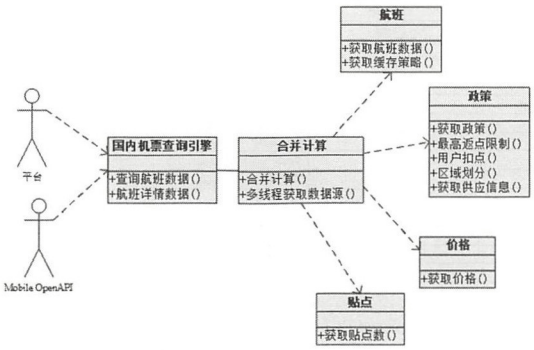

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 3. 领域图

下图是领域图，它从用例活动图演化而来，图中的行为与活动图有对应关系。平台或 Mobile 触发查询引擎后，多线程获取政策数据、贴点数据、价格数据和航班数据，然后进行合并计算。领域图是应用程序的业务逻辑模型，它的每一个框有可能是一个类，也可能是一个类库，或者是一个应用、一个子系统，它是可大可小、可伸缩、可扩展的。




### 4. 接口设计

什么是接口？接口是契约、连接和交互，它是应用与外部世界的联系者。有一位资深架构师说过，“我只需要设计好一套接口，让整个业务流转起来，我的工作就做完了，至于怎么实现我可以不知道”，这话有一定道理。以上契约遵循统一的 Request/Response 实现模式设计规范。接口设计案例如下图所示。

**3. 6. 1 查询航班数据接口**

- 接口名称：航班查询数据接口
- 接口描述：根据接入方的请求参数，合并计算并返回航班查询数据
- 接口提供者信息：国内机票查询引擎
- 接口使用者信息：平台、Moblie OpenAPI
- 接口方式：WCF
- 接口周期：实时

| 请求消息对象 | | | | | |
|---|---|---|---|---|---|
| 节点名 | 英文名称 | 中文名称 | 数据类型 | 必填 | 备注 |
| FlightSearchRequest | UserName | 用户名 | String | T | 用户名 |
| | RequestType | 请求类型 | String | T | 请求类型（OpenApi, Web） |
| | Scity | 出发城市 | String | T | 出发城市三字码(CSX) |
| | Ecity | 到达城市 | String | T | 到达城市三字码(CSX) |
| | RateFlag | 政策标记 | String | F | 获取政策 True 不获取政策 Flase(暂不提供) |
| | SDate | 出发日期 | String | T | 出发日期 yyyy-MM-dd |
| | AirLine | 航司二字码 | String | F | 航司二字码(CA) |
| | FlightNo | 航班号 | String | F | 航班号(MU1234) |
| | PolicyNum | 舱位返回政策条数 | Int | F | 返回政策条数，默认 1 个 |
| | Cabin | 舱位类型 | String | F | 舱位类型<br>A：所有舱位(默认)<br>B：经济舱<br>C：头等舱<br>D：全价舱<br>E：公务舱 |



仅供非商业用途或交流学习使用

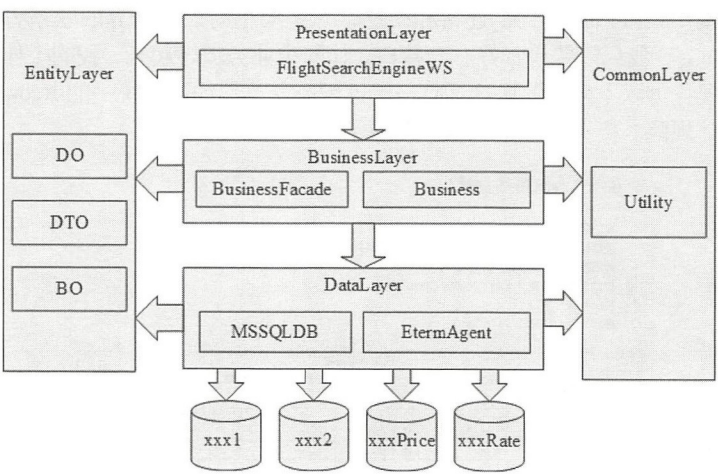
非卖品，仅供非商业用途或交流学习使用



| 响应消息对象 | | | | | |
|---|---|---|---|---|---|
| 节点名 | 英文名称 | 中文名称 | 数据类型 | 必填 | 备注 |
| FltSearchResponse | Success | 成功 | bool | T | 成功标志 |
| | ErrMsg | 错误信息 | String | F | 错误描述 |
| Flight | **Flights** | 航班集合 | List<Flight> | T | |
| | SCity | 出发城市 | String | T | PEK |
| | Ecity | 到达城市 | String | T | CSX |
| | SDate | 出发日期 | String | T | yyyy-MM-dd |
| | AirLine | 航司二字码 | String | T | MF |
| | FlightType | 机型 | String | T | 738 |
| | Stime | 起飞时间 | String | T | 06:00 |
| | Etime | 到达时间 | String | T | 08:20 |
| | Stop | 经停 | String | T | 0 |
| | Tax | 机建燃油 | String | T | 150 |
| | AirT | 航站楼 | String | T | T1,T2 |
| | **Cabins** | 舱位集合 | List<Cabin> | T | |
| Flight》Cabin | C | 舱位代码 | String | T | S |
| | CN | 舱位描述 | String | T | 经济舱 |
| | CL | 舱位等级 | String | T | B |
| | D | 折扣 | String | T | 40 |
| | P | 票面价 | String | T | 500 |
| | N | 舱位数量 | String | T | 0-9 A 大于 9 |
| | AlterNote | 变更说明 | String | F | 变更说明 |
| | EndNote | 改签说明 | String | F | 改签说明 |
| | RefundNote | 退票说明 | String | F | 退票说明 |
| | **Policys** | 政策集合 | List<Policy> | T | 政策集合 |

## 5. 分层设计

分层设计如下图所示。




## 6. 代码实现

下图（左）是第一个版本的代码实现，例如，SearchVerify 实现验证查询参数、CaculateBusiness 实现合并计算、PolicyBusiness 实现政策相关逻辑、PriceBusiness 实现价格相关逻辑、DiscountBusiness 实现贴点相关逻辑、CacheBusiness 实现缓存逻辑、UserBusiness 实现用户逻辑。下图（右）是第二个版本，相对第一个版本的实现要复杂一

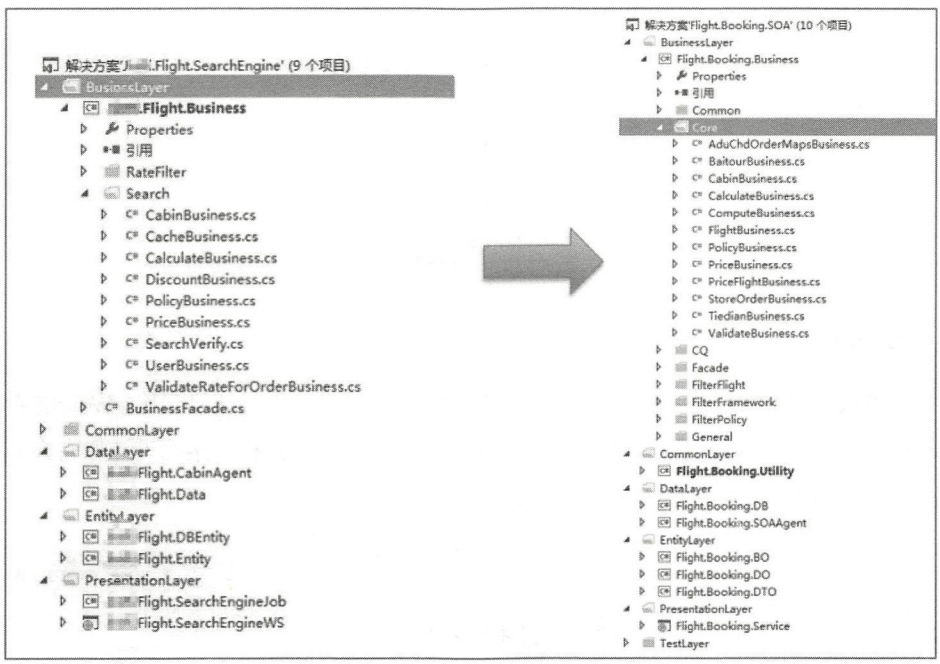


仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



些：ValidateBusiness 对应验证查询参数、CaculateBusiness 对应合并计算、PolicyBusiness 对应政策、PriceBusiness 对应价格、TiedianBusiness 对应贴点、FilterPolicy 对应政策过滤。可能读者已经发现，不管代码怎么升级改造，只要领域模型没有发生变化，业务模块就不会发生大的变化。




架构设计会改变编码方式，在架构设计阶段如果做好了领域模型，则可以在编码施工阶段先写业务逻辑层再写数据访问层。先定义好业务服务和数据接口定义，再根据数据定义来实现数据访问。这与表驱动的传统方式有所不同，先写数据层再写业务逻辑层，先写好数据表的增删改查，然后业务逻辑层只是简单地调用一下数据层，便提供给界面层使用。它只是一个简单代理，完全没有发挥业务逻辑层应有的价值。

### 7. 其他设计项

除了以上设计项，还有数据库设计、物理架构设计、非功能性设计。数据库设计有 E-R 图和表设计，物理架构设计有应用集群、应用部署图、域名等，非功能性设计有性能、可用性、伸缩性、扩展性、安全性等。最后是总结和表述，输出一份架构设计文档，详见下图和案例参考链接。



仅供非商业用途或交流学习使用

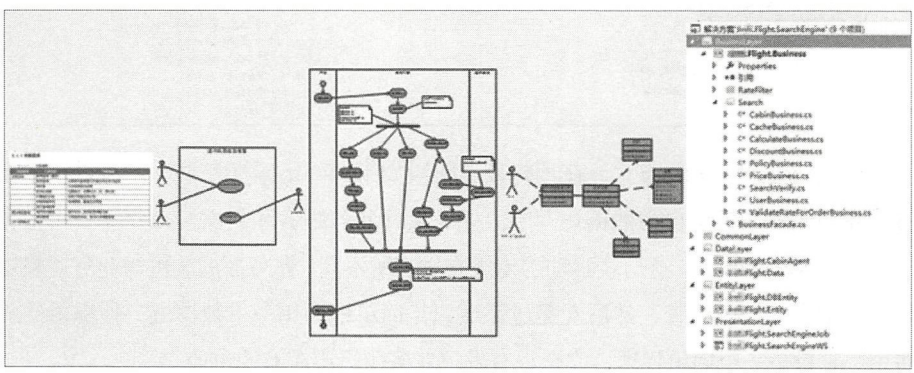
非卖品，仅供非商业用途或交流学习使用





### 8. 演化

下图是架构设计的关键过程，上一环是功能需求，下一环是代码实施，从功能到图纸到代码，从代码到图纸到功能，这是一个可演化可追溯的过程。架构设计如同施工图纸，能直接指导工程代码的实施，以及编码施工次序的改变。



功能需求→用例图→用例活动图→领域模型→代码实现

## 3.3 更多知识探讨

什么是探讨，什么是培训？培训是我有知识和经验，然后教给大家，我是正确的，大家照着干就可以了。而探讨是我有一个很好的问题，来问大家，来请教大家，以启发



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



你我的思维。接下来与读者一起探讨以下架构知识。

### 1. 关于设计表述

（1）一定要有架构设计文档吗？按教科书的要求是需要的，但真实的情况可能并不是这样的，没有设计文档的情况并不少见。

（2）架构设计文档要不要保存？项目做完后，它要保存多久呢？你待过的公司有保存吗？我们要沉下心来问问自己，追求真实比书本更重要。

（3）设计文档到底在为谁服务？为自己还是为别人？为半年后的自己，还是为公司或同事？

（4）设计可以省掉吗？在没有设计文档的前提下，是否可以编写高质量的代码？如果文档可以省掉，那么架构设计过程呢？

架构设计文档的编写并不简单，可能要花一周或一个月的时间，成本较高。设计表述方式有多种，具体如下。

- 架构设计文档：相当于造房子用的施工图纸，是相对比较正式的方式。
- 需求分析设计或项目排期会议：PRD 出来后，研发成员一起开需求分析设计会，分析讨论的过程也是设计的过程。或者通过项目排期，在估算项目难易程度和排期时，也有设计分析的成分。
- 设计邮件：一份简单的设计邮件，内容大约有问题描述、原因分析、技术方案和架构建议等。
- 非正式讨论：几个人站在白板前，讨论和画画，会议结束后再把讨论的内容拍照并发给参会人员。

### 2. 关于 UML

UML 是 Unified Modeling Language 的缩写，又称统一建模语言，始于 1997 年一个 OMG 标准。它是一个支持模型化和软件系统开发的图形化语言，为软件开发的所有阶段提供模型化和可视化支持。它不仅统一了 Booch、Rumbaugh 和 Jacobson 的表示方法，而且对其做了进一步的发展，并最终统一为标准建模语言。UML 图主要有用例图、时序图、活动图、类图、状态图、组件图和部署图。

UML 是设计表述和建模工具，虽然它的愿景是全生命周期，甚至用 UML 直接生成可执行软件。实际上这是很难的，不真正写代码，不可能明确所有细节。当然，UML 在



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

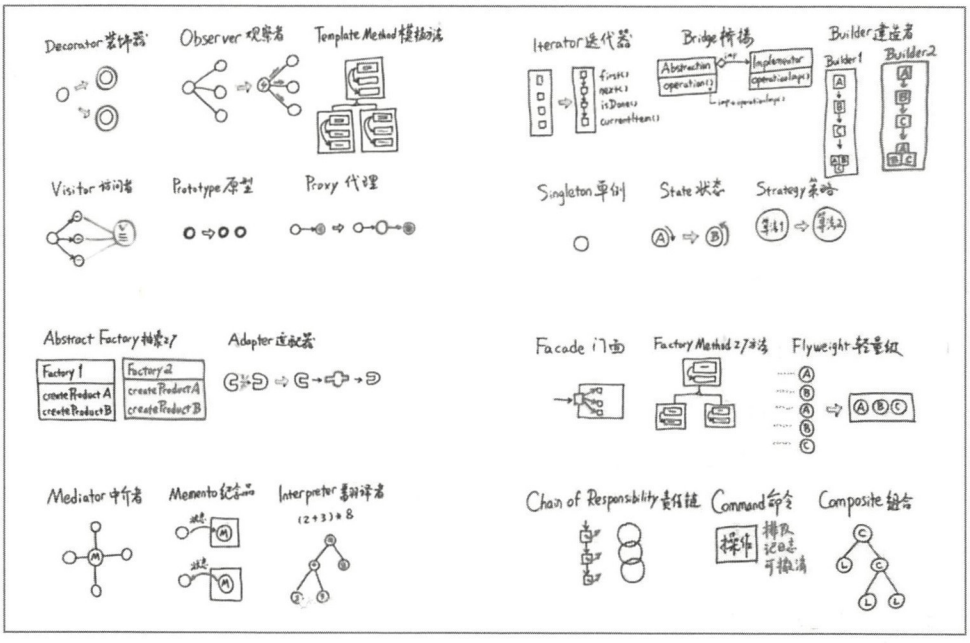


设计过程中还是有一定作用的，例如时序图、类图、状态图，这些如果不用 UML 图来表示而用文字来描述，则很难达成一致的理解。

UML 是理想建模工具吗？什么是理想的建模工具呢？船泊业 3D 建模，在未生产前就在计算机里把整个船构建起来。塑胶建模工具 ProE、商品房售楼部的沙盘，在未见实物前就可通过模型知道很多信息。理想建模工具应该是 3D 的、动态的、简单而形象的。UML 只是一种表述你头脑中思想的工具，相对而言，你头脑中的思想才重要。所选用的表述工具要根据双方的实际情况，简单清晰、利于沟通才是目的，并不一定就是 UML。

### 3. 关于设计模式

设计模式是一套被反复使用的、多数人知晓的、经过分类的代码设计经验的总结。使用设计模式是为了重用代码，让代码更容易被他人理解。设计模式使代码编制进一步工程化。每种模式都描述了一个不断重复发生的问题，以及该问题的核心解决方案，项目中合理地运用设计模式可以很好地解决很多问题。GoF的设计模式共有 23 种，理解意图是运用设计模式的关键，一图胜万言，下图是图解 23 种设计模式[1]。




1 图解23种设计模式图摘自：https://mp.weixin.qq.com/s/VhifkmoHZi_TdbWN3bbUPQ



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



设计模式是代码的形状，是代码结构设计的招式，是练功的套路，如同书是人类进步的阶梯。但练功是练功，打架是打架，真正的功夫要在大规模实战中所得。从设计模式到代码，再从代码重构到设计模式。设计模式不仅是设计出来的，也是重构“长”出来的。虽然重构并非一定会得到与设计模式完全相同的抽象结果，但重构是设计模式的迭代补充。设计模式如果使用得过早过多或不恰当，则会给代码增加不必要的结构复杂度。重构和设计模式的良好结合，使代码更趋于高品质和实用。GoF 设计模式是始于 1995 年的经典，主要解决当时软件的重用性、扩展性和可维护性问题。而在 20 多年后互联网时代的今天，版本迭代快、可随时在线更新，使用环境、语言和框架都已发生变化，许多模式是否还合时宜？设计模式在面试的时候用得多，还是在实际开发中用得多？可能每个人的答案都不一样，但每个工具都有其适用场景、收益和成本，思考这些问题有利于我们更好地使用它。

### 4. 关于设计原则（SOLID）

设计原则是设计模式的关键所在，原则和方法是决策的思想指南，设计原则（SOLID）具体如下。

- 单一职责原则（SRP）：一个类只承担一种类型的责任，一次只做一件事。
- 开闭原则（OCP）：对扩展开放，对修改关闭。开闭原则具有理想主义的色彩，它是面向对象设计的终极目标。
- 里氏代换原则（LSP）：任何基类可以出现的地方，子类一定可以出现，它是一个建议或约定。
- 接口隔离原则（ISP）：不能强迫用户去依赖那些他们不使用的接口。
- 依赖倒置原则（DIP）：高层模块不应该依赖低层模块，但它们都应该依赖抽象，客户第一。

### 5. 关于 DDD

DDD 是 Domain Driven Design 的缩写，翻译为领域驱动设计，它的核心是领域模型。什么是模型？装修人员从来没看过你的房子，但看过装修图后，就能知道你要装修成什么样。它的价值在于导航、精炼、统一表述。它能够帮助施工方和客户，全方面和多角度地看待问题，而不是盲人摸象。它有利于沟通、实现、维护和扩展。什么是领域？领

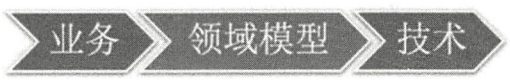


仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



是领地的意思，域是边界的意思。领域是一个专业科目，是人为的划分，一个领域一个边界一个框，领域会随着规模、角度和时代的变更而发生变化。例如，公司规模很小的时候，没有财务部，一个人既当会计，又当出纳。当公司规模变大一些时，可以一位做会计，另一位做出纳，可划分两个领域。当公司规模变得更大的时候，领域又变了，成立财务部，财务部里有多个人，每个人干的事情都不一样。业务在变，认知在变，领域的划分也要变。领域是主观的，它是对客观世界的阶段性认知。业务、领域模型与技术的关系如下图所示。




**业务问题→模型→技术解决**
**现实对象→抽象→技术对象**
**对象→类→对象**
**抽象与具体的引申→鸡与蛋**

领域模型处于业务问题与技术解决之间，先将业务对象抽象成领域模型，然后根据领域模型来实现技术对象。从对象到类再到对象，从具体到抽象再到具体，我们对抽象和具体再做进一步引申。请问，是先有鸡还是先有蛋？这个问题不好回答，给你一只具体的鸡和一个具体的蛋，你便能知道它们是父子关系、子父关系或没有关系，如果给你一只抽象的鸡和一个抽象的蛋，则是不知道它们是什么关系的。再请问，是先有类还是先有对象？这个问题也不好回答。在设计阶段，是先有对象再有类，在编码阶段，是先有类再有对象。整个过程是：架构师在设计阶段根据业务对象抽象出类，然后程序员在编码阶段，先编写类，再“New”出一个对象。从对象到类再到对象，从业务问题到领域模型再到技术解决方案，从问题域到领域模型再到代码实施，这是领域驱动的核心所在。**领域驱动设计=从问题域驱动领域模型构建+从领域模型驱动代码实施。**

DDD 的分层架构包括仓储层（Repository Layer）、领域层（Domain Layer）、应用层（Application Layer）、表现层（Presentation Layer）、基础设施层（Infrastructure Layer）。从仓库中取出原材料，然后流水线将人、材料、工具组织起来，最后输出给表现层。领域层不依赖于仓储层，而仓储层依赖于领域层。这相当于传统三层中业务逻辑层不依赖于数据层，而数据层依赖于业务逻辑层。为什么要这样呢？这是因为上层需要什么下层就提供什么，而不是下层有什么就提供什么，客户第一、按需生产都是这个道理。在技



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



术的具体实现上即依赖倒置（DIP），把接口放在上层，然后下层实现，最后使用 IoC 工具绑定即可。

### 6. 设计不足与过度设计

什么是设计不足，什么是过度设计？不能解决当前问题的就是设计不足；只能解决当前问题的是恰当设计；能解决当前问题，且又能解决未来一段时间问题的是良好设计；能解决当前问题，但面向未来设计过多，且成本较大，预测错误又不能解决未来问题的是过度设计。我们要追求恰当设计或良好设计，特别是互联网项目，变化大、迭代快，很难预测未来发生的事。

那什么是好的设计呢？好的设计是实用的、易于理解的，是谨慎克制的、简单的，是能够落地的、考虑施工成本的。好的设计要解决业务的问题，你的设计再好，但不能解决业务的问题，那么这个设计就是不好的设计。好的设计是谨慎克制的，不能为秀技术或个人意愿而过多使用复杂的技术。好的设计是能够落地的，如果你的设计在落地上出现很多问题，那么就是有问题的设计。没有人在设计时失败，只有实施时失败。

### 7. 架构设计是艺术

以上架构知识非常重要，但并不是知道了这些就能做好架构设计。这如同很多人都会画圆和直线，但并不会画画；很多人会使用钉板和菜刀，但并不能做一桌佳肴。

我们探讨一个具体的问题，“能异步的尽量异步”，互联网公司程序员经常说的这句话，是否正确？首先，程序员喜欢同步还是异步？用户喜欢同步还是异步？程序员为了并发量，会选择异步。用户不要等待，要求系统立即返回，会选择同步。那么在什么情况下使用同步，什么情况下使用异步呢？有几个考虑因素，第一个是复杂度，同步=异步+轮询/通知，同步相对简单，异步则相对复杂。第二个是可靠度，如果是 2/5/8 秒概率较大，那么最好选用同步。第三个是用户体验，当使用异步后，用户体验也需要改进，可立即返回给用户一个单号和进度条。第四个是业务成熟度，业务成熟度分萌芽期、发展期、成熟期和衰退期四个阶段。对于新业务，能同步就同步，当业务变得越来越成熟，访问量越来越大的时候，容易出现高并发量导致用户排队，这时异步是很好的选择。

在实际问题面前，选择同步还是异步？要看情况，需要经过分析和思考，你需要知道每一种选择的利弊。分析的过程往往比决策更重要，当你知道了每一种选择的利弊，



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



这时选择喜欢的就好，因为你只有喜欢了才能把事情做得更好。你的架构设计=你+架构设计，架构设计是科学，“你”是主观意识，最后的决策一定包含了你的个性和情感。科学发展到最后是艺术，架构设计也是艺术。

## 3.4 互联网公司的架构设计要怎么落地

互联网公司的架构设计是怎么做的呢？专职的架构师越来越少，架构部门也大都解散，为什么会这样，我们该怎么办？

### 1. 要不要做架构设计

哪些项目需要做架构设计呢？越大的项目越需要做架构设计，开发时间越长的项目越需要做架构设计，参与人员越多、内部越复杂、外部依赖越多、影响面越大、维护成本越高的项目越需要做架构设计。那互联网的项目呢？它有以下特征。

- 时间：开发的周期整体很长，可能维护 10 年、20 年，但单个应用的开发周期短，多半以天和周为单位。
- 规模：互联网项目整体很大，但单个应用规模小，会被拆分为多个小应用。
- 业务知识：为自己做系统，行业知识不缺，长期为一个系统服务，有时自己也是客户。
- 复杂度：研发人员多，内部关系复杂，外部依赖多，变化大、迭代快，在不断地演化，24 小时不间断运行。

### 2. MVP 与架构设计

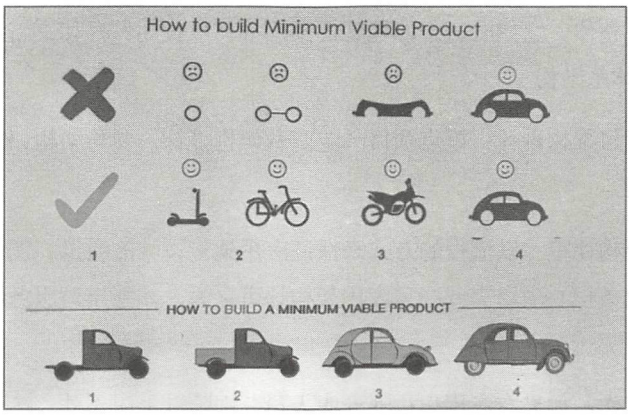
MVP 的英文全称是 Minimum Viable Product，是最小可行性产品的意思。如下图所示，用户需要一个交通工具，有两种实现方式，第一种做法是分多个阶段设计与制造，第一步是造一个轮子，第二步是造两个轮子，第三步是造一个盖子，第四步是造一辆可用的轿车。第二种做法是每一阶段都要满足用户从 A 地到 B 地的需求，第一步先造一个滑板，第二步是造一辆自行车，第三步是造一辆摩托车，第四步是造一辆轿车。第一个版本到第三个版本输出的产品都可以满足用户的基本需求，虽不完善，但可以解决用户的问题，并且越来越好，到了第四个版本的产品才满足客户预期。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






MVP 对架构设计提出了更高的要求。如果单纯从研发内部的角度考虑，则第一种是施工成本较低的方案，但我们需要以客户为中心，需要不断地满足客户的需求。所以在做设计时，不仅要考虑施工的成本，还要考虑客户需求、扩展性、继承性等，如上图第三种设计方案所示。

### 3. 互联网公司是怎么做的

互联网公司的架构设计的主流做法如下。

- 分工：将技术研发和业务研发相分离，下层是云平台部或基础架构部，提供 IaaS、PaaS 中间件等云服务，上层是各业务线的产品研发部，专注于业务场景的应用研发。
- 敏捷：业务研发敏捷化，产品人员与研发人员、测试人员实时沟通，以弥补行业知识的缺乏。
- 整体：技术委员会负责技术总体规划和技术成长。
- 未来：研究院解决未来的技术问题，如阿里达摩院、百度研究院。
- 应用架构：主要负责技术与业务的结合，由应用架构师、技术经理或高级程序员担当。

### 4. 应用架构要怎么落地

应用的架构设计要怎么落地，常见的方案如下。

（1）总体架构规划：手握地图，才能明确自己所处的位置。总体架构规划可以让每



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



个研发人员了解整体，它如同房子的地基框架图纸，可长期保存和更新维护。具体参考TOGAF开放组体系结构。

（2）单个项目架构设计：重点项目一定要做架构设计，参与架构评审，非重点项目可简化设计表述。

（3）应用架构评审：以流程的方式来保证应用架构设计的质量，例如，重构项目、跨部门项目、业务核心项目需要经过应用架构评审之后，才能申请服务器、数据库、域名等。

（4）其他工作：如果有应用架构师专职人员，则除以上工作外，还包括统一应用分层、制定代码规范、组织技术培训、中间件推广、应用性能调优等。

## 3.5 你给技术打个分

以上内容首先是一个启发性的问题，然后是初识架构设计，接着是一个真实的应用架构设计案例，并探讨了更多架构设计知识，包括设计表述、UML、设计模式、设计原则（SOLID）和DDD，最后是互联网公司的项目要怎么落地。这些知识点都是工具，这些工具到底怎么样呢？如果它是一个新知识，则我们不好妄加评价，但这些工具已经出来很多年了，我们也工作了这么多年。它好不好用，实不实用，我们都应该知道一些。现在，大家也当一次老板，请你给技术打个分，以下二维码是链接，欢迎进行绩效考评。


## 3.6 案例参考

- AppArchDemo案例参考地址：https://github.com/das2017/AppArchDemo



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 4 统一应用分层

## 4.1 为什么要统一应用分层

应用分层这件事情看起来很简单，但每个程序员都有自己的一套方法，哪怕是初学者。如何让一家公司的几百个应用采用统一的分层结构，并得到大部分程序员的认同呢？这可不是一件简单的事情，接下来通过真实案例与大家一起探讨，先问大家两个技术问题：

（1）服务的调用代码放到哪一层好呢？A 表现层；B 业务逻辑层；C 数据层；D 公共层。

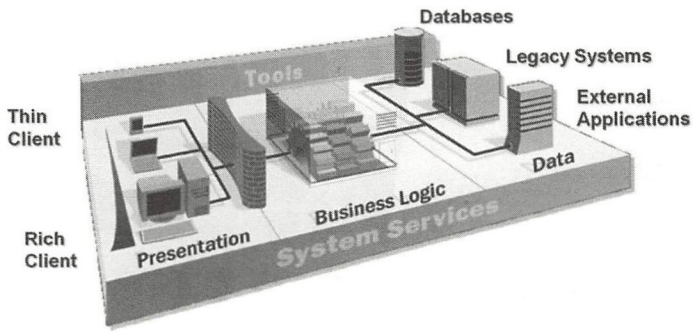
（2）如何组织好 VO（View Object，视图对象）、BO（Business Object，业务对象）、DO（Data Object，数据对象）、DTO（Data Transfer Object，数据传输对象）呢？

不同的人会有不同的答案，所以要统一公司应用分层，以减少开发维护学习成本，提高工作效率。统一应用分层要可大可小、简单易用、支持多种场景，我们采用 IPO 方式：I 是 Input、O 是 Output、P 是 Process。应用系统的本质是机器，是处理设备。统一应用分层示意图如下。

仅供非商业用途或交流学习使用

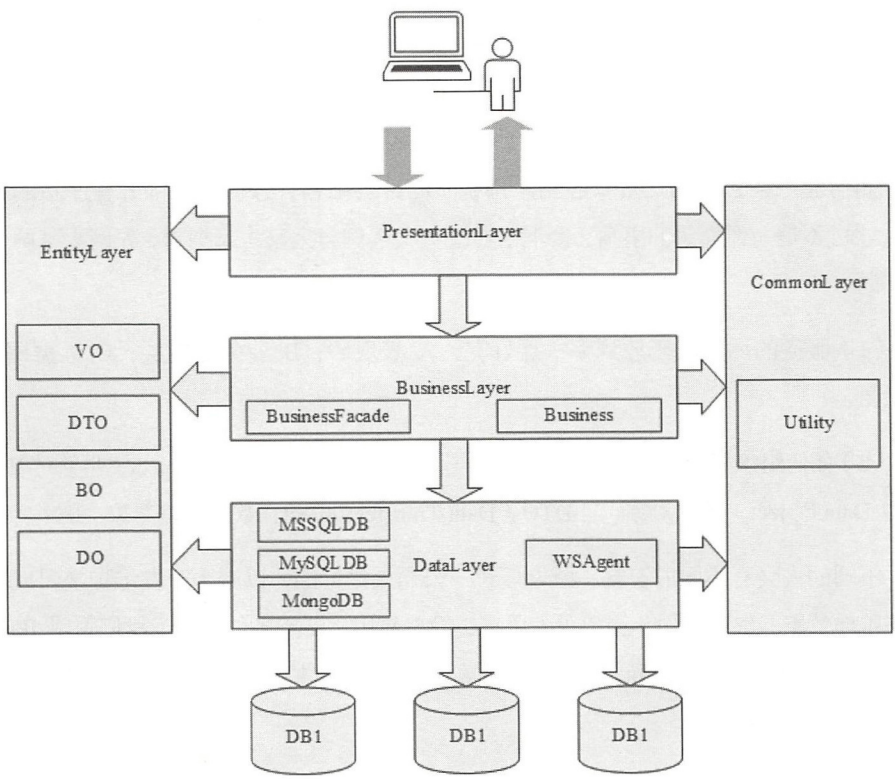
非卖品，仅供非商业用途或交流学习使用






## 4.2 统一应用逻辑架构

统一应用逻辑架构如下图所示。






仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



职责说明如下表所示。

| 层英文名 | 中文名 | 说明 |
| --- | --- | --- |
| PresentationLayer | 表现层 | 上层向用户提供服务，负责视图展示。项目类型包括 WebSite、WebForm、MVC、WCF、WebService 等 |
| BusinessLayer | 业务逻辑层 | 中间逻辑处理，负责应用系统的业务逻辑的处理 |
| DataLayer | 数据访问层 | 下层调用服务，负责数据资源提供方，如数据库、SOA、OpenAPI 的交互 |
| EntityLayer | 实体层 | VO：View Object，视图对象；DTO：Data Transfer Object，数据传输对象；BO：Business Object，业务对象；DO：Data Object，数据对象。在实际项目中，为简化设计可进行裁剪，BO 和 DO 为可选，DTO 属于服务项目类型，VO 属于网站项目类型，也不会同时存在 |
| CommonLayer | 公共层 | 工具类库，负责提供应用系统中常用的操作 |
| TestLayer | 测试层 | 单元测试（可选），负责对其他类库的自动化单元测试 |

- 文件夹分层法：应用分层采用文件夹方式的优点是可大可小、简单易用、统一规范，可以包括 5 个项目，也可以包括 50 个项目，以满足所有业务应用的多种不同场景。
- 调用规约：在开发过程中，需要遵循分层架构的约束，禁止跨层次的调用。
- 下层为上层服务：以用户为中心，以目标为导向。上层（业务逻辑层）需要什么，下层（数据访问层）就提供什么，而不是下层（数据访问层）有什么，就向上层（业务逻辑层）提供什么。
- 实体层规约：DO 是数据表对象，不是数据访问层对象，不是只能给数据访问层使用；DTO 是网络传输对象，不是表现层对象，不是只能给表现层使用；BO 是内存计算逻辑对象，不是业务逻辑层对象，不是只能给业务逻辑层使用。如果仅限定在本层访问，则导致单个应用内大量没有价值的对象转换。以用户为中心来设计实体类，可以减少无价值重复对象和无用转换。
- U 型访问：下行时表现层是 Input，业务逻辑层是 Process，数据访问层是 Output。上行时数据访问层是 Input，业务逻辑层是 Process，表现层是 Output。



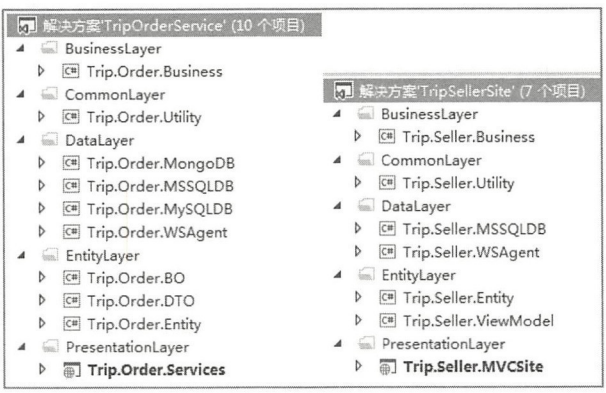
仅供非商业用途或交流学习使用

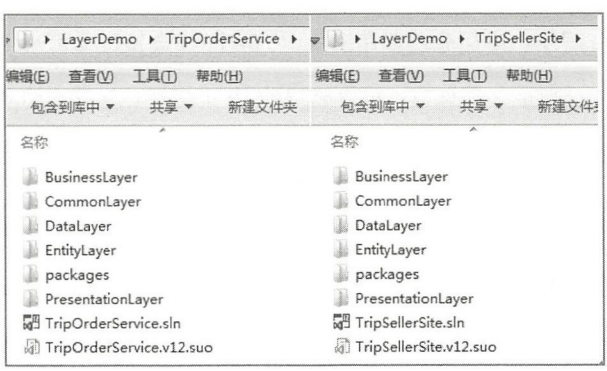
非卖品，仅供非商业用途或交流学习使用



## 4.3 分层规范实践

此规范我们用了四年，牵涉几百个应用，200 多个研发人员，是一个成功的实践。接下来就借用本章提供下载的 TripOrderService、TripSellerMVCSite 这两个 Demo 来进行具体规范的说明，以下是截图。




### 1．项目命名规则

项目命名规则：{产品线英文名全称}.{子系统英文名全称+应用名}.{项目职责英文名全称}，如 Trip.Seller.DTO。

### 2. 业务逻辑层规范

规范说明：

（1）项目名的命名规则：{产品线英文名全称}.{子系统英文名全称+应用名}.xxxBusiness，



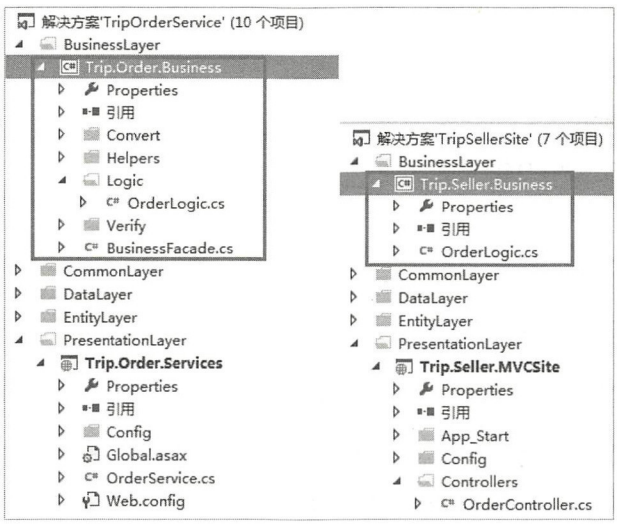
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



如下图中的 Trip.Order.Business。

（2）类名以 Logic 结尾，如下图中的 OrderLogic.cs。

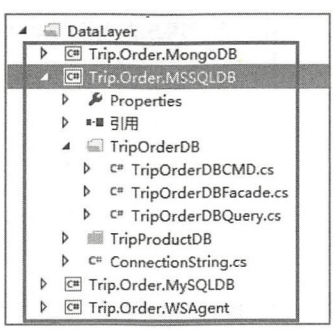



### 3. 数据操作层规范

规范说明：

（1）各数据操作项目名根据使用什么数据库进行分类，然后以 DB 为结尾，具体命名规则是{产品线英文名全称}.{子系统英文名全称+应用名}.{使用什么数据库}DB，如下图的 Trip.Order.MSSQLDB。

（2）如果涉及多个数据库访问，那么数据操作项目下的类文件需要按数据库名称（以 DB 为结尾）创建文件夹分开，如下图的 TripOrderDB 文件夹。






仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（3）建议在应用中使用 SQL 语句，不使用存储过程。在数据库中不新增存储过程，但旧的存储过程可以继续使用和修改。

（4）建议使用数据库（如 SQLServer）的最新特性进行分页，并将每个分页 SQL 直接写到应用中。

### 4. 实体层规范

- 数据传输对象 DTO 规范

规范说明：

（1）DTO 项目命名规则：{产品线英文名全称}.{子系统英文名全称+应用名}.DTO，如下图的 Trip.Order.DTO。

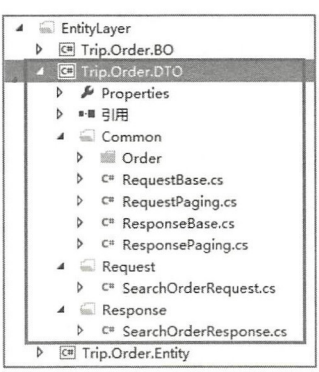
（2）请求参数 DTO 实体类、响应 DTO 实体类存放规范及其命名规则如下。

① 请求参数DTO实体类放在Request文件夹下，且命名规则为以Request结尾，如下图的 SearchOrderRequest.cs。

② 响应 DTO 实体类放在 Response 文件夹下，且命名规则为以 Response 结尾，如下图的 SearchOrderResponse.cs。

③ 如果请求参数 DTO 实体类或响应 DTO 实体类的属性中有对象或枚举，那么这些对象所属的类、枚举放在 DTO 项目的 Common 文件夹下。

（3）如果请求参数 DTO 实体类、响应 DTO 实体类有基类要继承，那么建议为基类取名为 RequestBase.cs、ResponseBase.cs，且这些基类直接放在 DTO 项目的 Common 文件夹下。






仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



- 视图对象 VO 规范

规范说明：

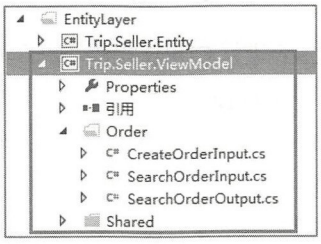
（1）VO 项目命名规则：{产品线英文名全称}.{子系统英文名全称+应用名}.ViewModel，如下图的 Trip.Seller.ViewModel。

（2）各 VO 实体类，我们用 Controller 名作为文件夹名进行分开，如下图的 Order 文件夹。

（3）VO 实体类名的命名建议：

① 请求参数 VO 实体类以 Input/Form/Query 结尾，如下图的 SearchOrderInput.cs。

② 响应 VO 实体类以 Output/List/Result 结尾，如下图的 SearchOrderOutput.cs。

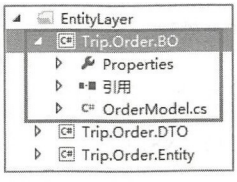



- 业务对象 BO 规范（可选）

规范说明：

（1）BO 项目命名规则：{产品线英文名全称}.{子系统英文名全称+应用名}.BO，如下图的 Trip.Order.BO。

（2）以 Model 结尾，如下图的 OrderModel.cs。




（3）为了简化设计，BO 项目为可选，可在 DO 项目里建文件夹。

- 数据对象 DO 规范（可选）



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

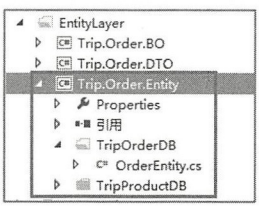


规范说明：

（1）DO 项目命名规则：{产品线英文名全称}.{子系统英文名全称+应用名}.Entity，如下图的 Trip.Seller.Entity。

（2）如果涉及多个数据库访问的，那么需要按数据库名称（以 DB 为结尾）创建文件夹，如下图的 TripOrderDB 文件夹。

（3）表名+Entity 结尾，如下图的 OrderEntity.cs。




（4）DO 是数据表对象，供单表进行 CURD 操作。对于多表查询请求对象和返回对象，可定义新对象或使用现有对象（DTO/BO）来完成。

### 5. 数据库连接配置规范

数据库连接配置样例如下图所示。

```
Database.Dev.config
1  <?xml version="1.0" encoding="utf-8"?>
2  <connectionStrings>
3    <add name="TripOrderDB_SELECT" connectionString="MO74cB4ekCz/kIFnIbx7PgM4K79PfGCn77Yag5dGhk7J1EcEpFTLO5bX2ryVAOvJLPPSfU4eECDvvSKjIZWS
     +NgkAx5JFAOamVprCaAJYrCSQWo9MoZkjCX4QTcDy2o8ASCotkoMdDKOPALOaFACeqLp2kOWKA4s" providerName="System.Data.SqlClient"/>
4    <add name="TripOrderDB_INSERT" connectionString="MO74cB4ekCz/kIFnIbx7PgM4K79PfGCn77Yag5dGhk7J1EcEpFTLO5bX2ryVAOvJLPPSfU4eECDvvSKjIZWS
     +NgkAx5JFAOamVprCaAJYrCSQWo9MoZkjCX4QTcDy2o8ASCotkoMdDKOPALOaFACeqLp2kOWKA4s" providerName="System.Data.SqlClient"/>
5  </connectionStrings>
```

规范说明：

（1）数据库连接的配置必须读写分离。

（2）数据库连接字符串建议加密处理。

（3）数据库连接配置名的命名规则：{以 DB 为结尾的数据库名称}_读写类型，如 TripOrderDB_SELECT、TripOrderDB_INSERT。

### 6. 配置文件规范

规范说明：



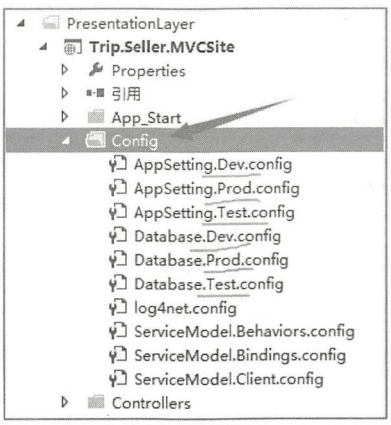
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（1）所有配置文件（除 Web.config 文件外）都必须放到 Config 文件夹下。

（2）所有配置文件（除 Web.config 文件外）按不同环境区分开，具体命名规则是{功能模块英文名}.{环境英文简称名}.config，其中本地环境的英文简称是 Dev，测试环境的英文简称是 Test，正式环境的英文简称是 Prod，如下图的 AppSetting.Dev.config。




（3）保持 Web.config 配置文件的干净，只留环境设置节点，如下图所示。

Web.config

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  有关如何配置 ASP.NET 应用程序的详细信息，请访问
  http://go.microsoft.com/fwlink/?LinkId=301880
  -->
<configuration>
  <appSettings configSource="Config\AppSetting.Dev.config" />
  <connectionStrings configSource="Config\Database.Dev.config" />

  <system.serviceModel>
    <bindings configSource="Config\ServiceModel.Bindings.config" />
    <behaviors configSource="Config\ServiceModel.Behaviors.config" />
    <client configSource="Config\ServiceModel.Client.config" />
  </system.serviceModel>
```

### 7. 静态资源文件规范

规范说明：

（1）公共的静态资源文件（CSS、JS、Image 等）放在另外的静态站点中，统一由前端进行开发和维护。一般 CSS 文件放在 css 文件夹下，JS 文件放在 js 文件夹下，Image 图片文件放在 img 文件夹下，如下图的左半部分所示。

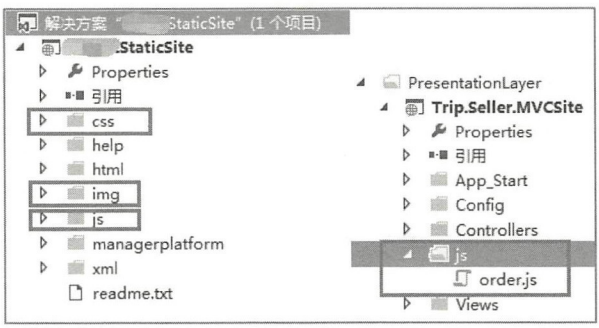


仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（2）与某项业务有关的 JS 文件可以放到各自业务项目的表现层（PresentationLayer）下，以方便开发人员调试，JS 文件可放在项目的 js 文件夹下，如下图的右半部分所示。




（3）静态资源文件必须使用版本号管理，以免更新后由于客户端浏览器缓存而导致站点使用的依然是旧版本的静态资源文件：

```
<script src="~/js/order.js?v=@AppSetting.StaticFileVersion"></script>
```

（4）可以采用前后端完全分离的方式，让 Java 或.NET 开发资源撤出表现层，以专注于业务逻辑需求的迭代。

## 4.4 互动问答

**问：**服务的调用代码应该放到哪一层呢？

A 表现层、B 业务逻辑层、C 数据层、D 公共层。

**答：**我们的规范统一放到数据资源访问层。上层提供服务，下层调用服务，中间处理业务逻辑。

**问：**如何组织好 VO（View Object，视图对象）、BO（Business Object，业务对象）、DO（Data Object，数据对象）、DTO（Data Transfer Object，数据传输对象）呢？

**答：**通常有两种做法，限定访问范围和不限定访问范围，实际项目中可根据需要选择、折中或裁剪。我们使用后者，将 EntityLayer 作为通用对象放到左侧，具体可参考实体层规约。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



**问：**应用分层范例代码的编写需要注意些什么？

**答：**应用分层范例的代码要想写好，非常不容易，很容易引起争议，很难让所有人满意。我们在具体实践时应遵循以下几点。

（1）应用分层范例的主要价值是明确层的职责和交互方式，每个层的职责是什么，哪些要干，哪些不要干，以及层与层之间的依赖关系和交互方式。

（2）私人定制：减少通用帮助类的编写，如果每一个应用中有大量相同的帮助类，则在架构层面上是有问题的。在我们的几百个线上应用中，尽量减少通用的代码，包括分页帮助类、数据库帮助类、缓存帮助类、MQ 帮助类、日志帮助类、AOP 帮助类和线程帮助类。业务应用的重点是为业务服务，每一个应用都是特别的，都需要私人定制，极少有通用的代码，如果有，那么应该由框架或组件专门解决。

（3）少即是多：应用的场景多，参考人员多，每个人的想法不同，牵涉的时间长，所以尽量只做大家都认同的规范、正确的事情，要自下向上、减少有争议的代码范例，否则一个错误会放大百倍，一个有争议的规范将会很难推行。

（4）追求简单：代码编写可分为三个层次，简单、复杂、简单。第一个简单是不知道的简单，第二个复杂是知道后的复杂，第三个简单是知道后有取舍的简单。范例代码要追求简单，既可轻松扩展以支持复杂场景，又要简单到初级程序员也能操作。

（5）内聚大于解耦：内聚是指部门内有共同的目标，然后大家紧密合作。解耦是指部门间各自职责明确，然后减少不必要的连接。一个应用如同一个部门，应该有一个共同的目标和职责，然后大家紧密合作。换句话说，应用内部应减少不必要的契约接口（如同公司间合作才签合同），减少不必要的依赖注入实现，减少不必要且代价过大的解耦。一切以简单实用为主，以应用价值输出、应用的目标（接口或界面）为导向。

## 4.5 Demo 下载

- LayerDemo 下载地址：https://github.com/das2017/LayerDemo



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 5 生产环境诊断工具 WinDbg

生产环境偶尔会出现一些异常问题，WinDbg 或 GDB 是解决此类问题的利器。调试工具 WinDbg 如同医生的听诊器，是系统“生病”时进行诊断的逆向分析工具。Dump 文件类似于飞机的黑匣子，记录生产环境程序运行的状态。本章主要介绍调试工具 WinDbg 和抓包工具 ProcDump 的使用，并分享一个真实的案例。多年前不知谁写的代码，导致每一两个月偶尔出现 CPU 飙高的现象。我们先使用 ProcDump 在生产环境中抓取异常进程的 Dump 文件，然后在不了解代码的情况下通过 WinDbg 命令进行分析，最终定位有问题的那行代码。

## 5.1 诊断工具简介

### 1. WinDbg

WinDbg 是运行在 Windows 平台下的、强大的用户态和内核态调试工具。相比较于 Visual Studio（VS），它是一个轻量级的调试工具，所谓轻量级指的是它的安装文件较小，但是其调试功能却比 VS 更强大。它的另外一个用途是用来分析 Dump 数据。WinDbg 是

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



Microsoft 公司免费调试集合中的 GUI 的调试器，支持 Source 和 Assembly 两种模式的调试。WinDbg 不仅可以调试应用程序，还可以进行 Kernel Debug。结合 Microsoft 的 Symbol Server，可以获取系统符号文件，便于应用程序和内核的调试。WinDbg 支持的平台包括 x86、IA64、AMD64。虽然 WinDbg 也提供图形界面操作，但它最出色的地方还是功能强大的调试命令，一般情况会结合 GUI 和命令行进行操作，常用的视图有局部变量、全局变量、调用栈、线程、命令、寄存器、白板等。其中“命令”视图是默认打开的。

### 2. DebugDiag

DebugDiag 最初是为了帮助分析 IIS 的性能问题而开发的，它同样可以用于任何其他的进程。DebugDiag 工具主要用于帮助解决如挂起、 速度慢、 内存泄漏或内存碎片，以及任何用户模式进程崩溃等问题。该工具包括附加调试脚本、侧重于互联网信息服务( IIS ) 应用程序、Web 数据访问组件、COM+和相关 Microsoft 技术、SharePoint 和.NET。它提供可扩展对象模型中的 COM 对象的形式，并具有一个内置的报告框架提供的脚本主机。它由 3 部分组成，包括调试服务、调试器主机和用户界面。

### 3. ProcDump

ProcDump 是 System Internal 提供的一个专门用来监测程序 CPU 高使用率从而生成进程 Dump 文件的工具。ProcDump 可以根据系统的 CPU 使用率或指定的性能计数器来针对特定进程生成一系列的 Dump 文件，以便调试者对事故原因进行分析。

## 5.2 获取异常进程的 Dump 文件

有以下四种方式获取 Dump 文件，具体如下。

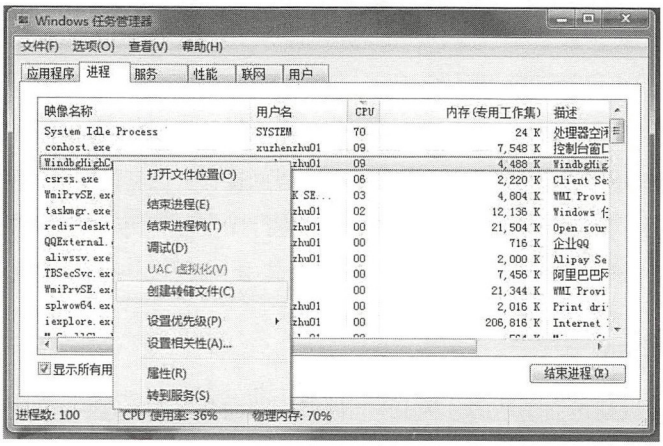
（1）通过“任务管理器”的“创建转储文件”获取 Dump 文件，如下图所示，这样获取的是 MinDump。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






（2）利用 WinDbg 的 adplus 获取 Dump 文件，如下图所示，这样获取的是 FullDump。

```
C:\Users'        >cd C:\Program Files\Debugging Tools for Windows (x64)

C:\Program Files\Debugging Tools for Windows (x64)>adplus -hang -pn WindbgHighCp
u.exe -o E:\dumps
Starting ADPlus
***************************************************************
*                                                             *
* ADPlus Flash V 7.01.002 02/27/2009                          *
*                                                             *
*   For ADPlus documentation see ADPlus.doc                   *
*   New command line options:                                 *
*     -pmn <procname> - process monitor                       *
*        waits for a process to start                         *
*     -po <procname> - optional process                       *
*        won't fail if this process isn't running             *
*     -mss <LocalCachePath>                                   *
*        Sets Microsoft's symbol server                       *
*     -r <quantity> <interval in seconds>                     *
*        Runs -hang multiple times                            *
*                                                             *
*   ADPlusManager - an additional tool to facilitate          *
*   the use of ADPlus in distributed environments like        *
*   computer clusters.                                        *
*   Learn about ADPlusManager in ADPlus.doc                   *
*                                                             *
***************************************************************

Attaching to 17976 - WindbgHighCpu in Hang mode 05/04/2017 00:04:05
Logs and memory dumps will be placed in E:\dumps\20170504_000405_Hang_Mode

C:\Program Files\Debugging Tools for Windows (x64)>_
```

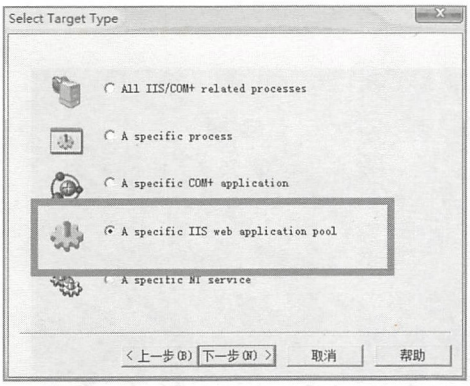
（3）通过 DebugDiag 创建.NET 异常转储 Dump 文件，如下图所示。

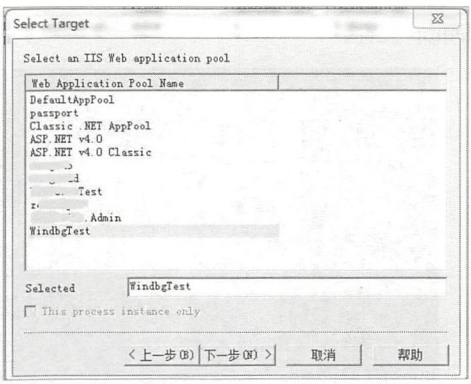


仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






（4）通过 ProcDump 抓取异常线程 Dump 文件。

现在重点介绍通过 ProcDump 抓取异常线程 Dump 文件，使用方法如下。

① 命令行：

```
procdump [-a] [[-c|-cl CPU usage] [-u] [-s seconds]] [-n exceeds] [-e
[1 [-b]] [-f <filter,...>] [-g] [-h] [-l] [-m|-ml commit usage] [-ma | -m
p] [-o] [-p|-pl counter threshold] [-r] [-t] [-d <callback DLL>] [-64] <
[-w] <process name or service name or PID> [dump file] | -i <dump file> |
-u | -x <dump file> <image file> [arguments] >] [-? [ -e]
```

② 实例：

```
procdump -c 70 -s 5 -ma -n 3 w3wp
```

当系统 CPU 使用率持续 5 秒超过 70%时，连续抓 3 个 Full Dump。

```
procdump outlook -p "\Processor(_Total)\% Processor Time" 80
```

当系统 CPU 使用率超过 80%时，抓取 Outlook 进程的 Mini Dump。

```
procdump -ma outlook -p "\Process(Outlook)\Handle Count" 10000
```

当 Outlook 进程 Handle 数超过 10000 时，抓取 Full Dump。

```
procdump -ma 4572
```

直接生成进程号为 4572 的 Full Dump。

下图是在 WindgbHighCpu 进程中造成 High CPU 时运行 ProcDump 命令的效果，可以看到在 CPU 每次持续 5 秒达到 5%后就会生成相应的 Dump 文件，共生成了 3 份 Full



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



Dump 文件。

```
D:\Procdump>procdump -c 5 -s 5 -ma -n 3 WindbgHighCpu

ProcDump v7.1 - Writes process dump files
Copyright (C) 2009-2014 Mark Russinovich
Sysinternals - www.sysinternals.com
With contributions from Andrew Richards

Process:               WindbgHighCpu.exe (13496)
CPU threshold:         >= 5% of system
Performance counter:   n/a
Commit threshold:      n/a
Threshold seconds:     5
Hung window check:     Disabled
Log debug strings:     Disabled
Exception monitor:     Disabled
Exception filter:      *
Terminate monitor:     Disabled
Cloning type:          Disabled
Concurrent limit:      n/a
Avoid outage:          n/a
Number of dumps:       3
Dump folder:           D:\Procdump\
Dump filename/mask:    PROCESSNAME_YYMMDD_HHMMSS


Press Ctrl-C to end monitoring without terminating the process.

[13:42:27] CPU: 5% 1s
[13:42:28] CPU: 8% 2s
[13:42:29] CPU: 8% 3s
[13:42:30] CPU: 8% 4s
[13:42:31] CPU: 6% 5s (Trigger)
[13:42:31] Dump 1 initiated: D:\Procdump\WindbgHighCpu.exe_150624_134231.dmp
[13:42:32] Dump 1 writing: Estimated dump file size is 64 MB.
[13:42:33] Dump 1 complete: 64 MB written in 2.0 seconds
[13:42:35] CPU: 7% 1s
[13:42:36] CPU: 5% 2s
[13:42:37] CPU: 5% 3s
[13:42:38] CPU: 10% 4s
[13:42:39] CPU: 9% 5s (Trigger)
[13:42:39] Dump 2 initiated: D:\Procdump\WindbgHighCpu.exe_150624_134239.dmp
[13:42:40] Dump 2 writing: Estimated dump file size is 64 MB.
[13:42:40] Dump 2 complete: 64 MB written in 0.3 seconds
```

③ 注意：

- ProcDump 需要进程已经启动，并且中途不能停止。比如需要抓取 IIS Worker Process 的 High CPU Dump，由于 IIS Worker Process 默认会配置 Idle Timeout = 20 min，即该进程在 20 分钟内没有任何请求就会自动结束，这种情况下 ProcDump 也会自动结束，需要重新运行命令。因此如果目标程序存在这样的配置，则需要暂时将该配置取消。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



- 有些系统管理员希望能够运行该工具后退出用户 session，ProcDump 是做不到的，如果有这种需求则可以考虑使用 DebugDiag。
- 在调试 High CPU 问题的时候经常用到的一个命令是!runaway，但是有些时候!runaway 在 ProcDump 抓取 Dump 文件的过程中运行不起来，报错信息如下：

```
   0:000> !runaway ERROR: !runaway: extension exception 0x80004002. "Unab
le to get thread times - dumps may not have time information"
```

解决的方法是将 Debugging Tools for Windows（WinDbg）安装目录下的 dbghelp.dll 复制到 procdump.exe 所在目录下，然后运行命令抓取 Dump。

## 5.3 WinDbg 的使用方法

操作步骤如下。

### 1. 抓取异常程序的 Dump 文件

### 2. 设置符号表

符号表是 WinDbg 关键的“数据库”，如果没有它，则 WinDbg 基本上无法分析更多问题。所以使用 WinDbg 之前设置符号表，是必须要执行的一步。

（1）运行 WinDbg，然后按“Ctrl+S”快捷键弹出符号表设置窗。

（2）将符号表地址 SRV*C:\Symbols*http://msdl.microsoft.com/download/symbols 粘贴在输入框中，单击确定即可。单击确定之前，请先确认“C:\Symbols”文件夹是否已被新建。

> 注：“C:\Symbols”表示符号表本地存储路径，建议固定路径，可避免符号表重复下载。

### 3. 打开第一个 Dump 文件

当获取一个 Dump 文件后，可使用“Ctrl+D”快捷键来打开一个 Dump 文件，或者单击 WinDbg 界面上的“File”→“Open Crash Dump...”按钮来打开一个 Dump 文件。第一次打开 Dump 文件时，可能会收到如下图所示的提示，出现这个提示时，勾选“Don’t ask

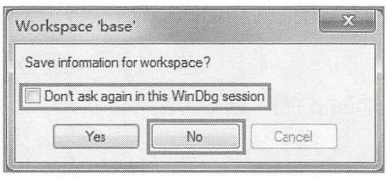


仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



again in this WinDbg session”，然后单击“No”按钮即可。




当你想打开第二个 Dump 文件时，可能因为上一个分析记录未清除，导致无法直接分析下一个 Dump 文件，此时可以使用“Shift+F5”快捷键关闭上一个对 Dump 文件的分析记录。

### 4. 通过简单的几个命令来分析 Dump 文件

分享一个数据库连接超时的 Dump 案例的分析过程。

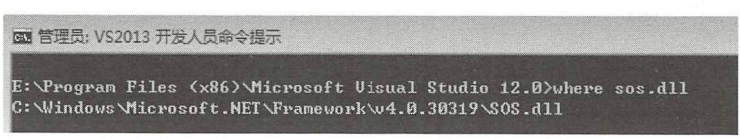
当打开一个 Dump 文件后，可能因为信息太多，让你无所适从，不过没关系，我们只需要关注几个关键信息就可以了。

（1）加载 SOS 扩展命令。

加载 SOS 之前，先确定 SOS 的位置和版本，确定方法如下。

如果安装了 Visual Studio，那么打开 VS 的命令行：Visual Studio Tools→VS2013 开发人员命令提示→以管理员身份运行。

然后在打开的 VS 命令行中输入 where sos.dll，获得 SOS 的位置和版本号，如下图所示。




确定 SOS 位置和版本号后，开始加载 SOS 扩展命令：

```
.load C:\Windows\Microsoft.NET\Framework64\v4.0.30319\SOS.dll
```

结果如下图所示。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



```
Loading Dump File [E:\dumps\w3wp__WindbgTest__PID__5140__Date__07_03_2015__Time_11_32_18AM__278__Firs
User Mini Dump File with Full Memory: Only application data is available

Comment: 'Dump created by DbgHost. First Chance CLR (.NET) 4.0 Exception - All Exception Types'

************* Symbol Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       srv*c:\symbols*http://msdl.microsoft.com/download/syml
Symbol search path is: srv*c:\symbols*http://msdl.microsoft.com/download/symbols
Executable search path is:
Windows 7 Version 7601 (Service Pack 1) MP (4 procs) Free x64
Product: WinNt, suite: SingleUserTS
Built by: 6_1_7601.18015 (win7sp1_gdr.121129-1432)
Machine Name:
Debug session time: Fri Jul  3 11:32:18.000 2015 (UTC + 8:00)
System Uptime: 0 days 1:54:47.704
Process Uptime: 0 days 0:01:33.000
.....................................................................
.....................................................................
.........
Loading unloaded module list
.
This dump file has an exception of interest stored in it.
The stored exception information can be accessed via .ecxr.
(1414.724): CLR exception - code e0434352 (first/second chance not available)
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for KERNELBASE.dll -
KERNELBASE!RaiseException+0x3d:
000007fe`fce8b3dd 4881c4c8000000  add     rsp,0C8h
0:031> .load  C:\Windows\Microsoft.NET\Framework64\v4.0.30319\SOS.dll
```

（2）使用 !clrstack 命令查看当前的调用堆栈信息，如下图所示。

```
0:031> !clrstack
OS Thread Id: 0x724 (31)
        Child SP               IP Call Site
000000000d7baf08 000007fefce8b3dd [HelperMethodFrame: 000000000d7baf08]
000000000d7baff0 000007fedb1df45a *** WARNING: Unable to verify checksum for System.Data.ni.dll
System.Data.SqlClient.SqlInternalConnectionTds.LoginNoFailover(System.Data.SqlClient.ServerInfo, System.String, System.Security.SecureStr
000000000d7bced8 000007feee7a7b15 [HelperMethodFrame: 000000000d7bced8]
000000000d7bcfc0 000007fedb1e2966 System.Data.SqlClient.SqlInternalConnection.OnError(System.Data.SqlClient.SqlException, Boolean, System
000000000d7bd010 000007fedab72411 System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(System.Data.SqlClient.TdsParserStateObject, Bo
000000000d7bd0d0 000007fedb1dfb58 System.Data.SqlClient.TdsParser.Connect(System.Data.SqlClient.ServerInfo, System.Data.SqlClient.SqlInte
000000000d7bd190 000007fedab6982b System.Data.SqlClient.SqlInternalConnectionTds.AttemptOneLogin(System.Data.SqlClient.ServerInfo, System
000000000d7bd220 000007fedab68bab System.Data.SqlClient.SqlInternalConnectionTds.LoginNoFailover(System.Data.SqlClient.ServerInfo, System
000000000d7bd3c0 000007fedab6870d System.Data.SqlClient.SqlInternalConnectionTds.OpenLoginEnlist(System.Data.ProviderBase.TimeoutTimer, S
000000000d7bd470 000007fedab6834e System.Data.SqlClient.SqlInternalConnectionTds..ctor(System.Data.ProviderBase.DbConnectionPoolIdentity,
000000000d7bd510 000007fedab67f7a System.Data.SqlClient.SqlConnectionFactory.CreateConnection(System.Data.Common.DbConnectionOptions, Sys
000000000d7bd620 000007fedab67a38 System.Data.ProviderBase.DbConnectionFactory.CreatePooledConnection(System.Data.ProviderBase.DbConnecti
000000000d7bd690 000007fedab677f5 System.Data.ProviderBase.DbConnectionPool.CreateObject(System.Data.Common.DbConnection, System.Data.Com
000000000d7bd750 000007fedab67547 System.Data.ProviderBase.DbConnectionPool.UserCreateRequest(System.Data.Common.DbConnection, System.Dat
000000000d7bd7a0 000007fedab66c5c System.Data.ProviderBase.DbConnectionPool.TryGetConnection(System.Data.Common.DbConnection, UInt32, Boo
000000000d7bd880 000007fedab66954 System.Data.ProviderBase.DbConnectionPool.TryGetConnection(System.Data.Common.DbConnection, System.Thre
000000000d7bd910 000007fedab66401 System.Data.ProviderBase.DbConnectionFactory.TryGetConnection(System.Data.Common.DbConnection, System.T
000000000d7bd9d0 000007fedab65d8b System.Data.ProviderBase.DbConnectionInternal.TryOpenConnectionInternal(System.Data.Common.DbConnection
000000000d7bda30 000007fedab63809 System.Data.SqlClient.SqlConnection.TryOpenInner(System.Threading.Tasks.TaskCompletionSource`1<System.D
000000000d7bdad0 000007fedab636c6 System.Data.SqlClient.SqlConnection.TryOpen(System.Threading.Tasks.TaskCompletionSource`1<System.Data.P
000000000d7bdb4f 000007fedab6[illegible] System.Data.SqlClient.SqlConnection.Open()
DapperDemo.Utils.Common.OpenConnection() [E:\work\project\windbgTest\Utils\Common.cs @ 16]
000000000d7bdc10 000007fe8f0223a8 DapperDemo.QueryDemo.GetCitys() [E:\work\project\windbgTest\QueryDemo.cs @ 148]
000000000d7bdcd0 000007fe8f0222a4 windbgTest.test.BtnHighCPU_Click(System.Object, System.EventArgs) [E:\work\project\windbgTest\test.aspx
000000000d7bdf50 000007fedaa58dcb *** WARNING: Unable to verify checksum for System.Web.ni.dll
System.Web.UI.WebControls.Button.RaisePostBackEvent(System.String)
000000000d7bde60 000007[illegible]ege.ProcessRequest(Boolean, Boolean)
000000000d7bded0 000007fedaa2c367 System.Web.UI.Page.ProcessRequest()
000000000d7bdf70 000007fedaa2a9c3 System.Web.UI.Page.ProcessRequest(System.Web.HttpContext)
000000000d7bdfc0 000007fe8f0220f3 *** WARNING: Unable to verify checksum for App_Web_ss5jslja.dll
ASP.test_aspx.ProcessRequest(System.Web.HttpContext)
000000000d7bdff0 000007fedaa356b1 System.Web.HttpApplication+CallHandlerExecutionStep.System.Web.HttpApplication.IExecutionStep.Execute()
000000000d7be0d0 000007fede9fb305 System.Web.HttpApplication.ExecuteStep(IExecutionStep, Boolean ByRef)
000000000d7be170 000007fedaa1931a System.Web.HttpApplication+PipelineStepManager.ResumeSteps(System.Exception)
000000000d7be2c0 000007fede9fb523 System.Web.HttpApplication.BeginProcessRequestNotification(System.Web.HttpContext, System.AsyncCallback
000000000d7be310 000007fede9f527e System.Web.HttpRuntime.ProcessRequestNotificationPrivate(System.Web.Hosting.IIS7WorkerRequest, System.W
000000000d7be3b0 000007fede9fe4e1 System.Web.Hosting.PipelineRuntime.ProcessRequestNotificationHelper(IntPtr, IntPtr, IntPtr, Int32)
000000000d7be5c0 000007fede9fdea2 System.Web.Hosting.PipelineRuntime.ProcessRequestNotification(IntPtr, IntPtr, IntPtr, Int32)
000000000d7be610 000007fedf168ab1 DomainNeutralILStubClass.IL_STUB_ReversePInvoke(Int64, Int64, Int64, Int32)
000000000d7bedf8 000007feee662ae8 [InlinedCallFrame: 000000000d7bedf8] System.Web.Hosting.UnsafeIISMethods.MgdIndicateCompletion(IntPtr,
000000000d7bedf8 000007fedeaab84b [InlinedCallFrame: 000000000d7bedf8] System.Web.Hosting.UnsafeIISMethods.MgdIndicateCompletion(IntPtr,
000000000d7bedd0 000007fedeaab84b DomainNeutralILStubClass.IL_STUB_PInvoke(IntPtr, System.Web.RequestNotificationStatus ByRef)
000000000d7beea0 000007fede9fe6cf System.Web.Hosting.PipelineRuntime.ProcessRequestNotificationHelper(IntPtr, IntPtr, IntPtr, Int32)
```

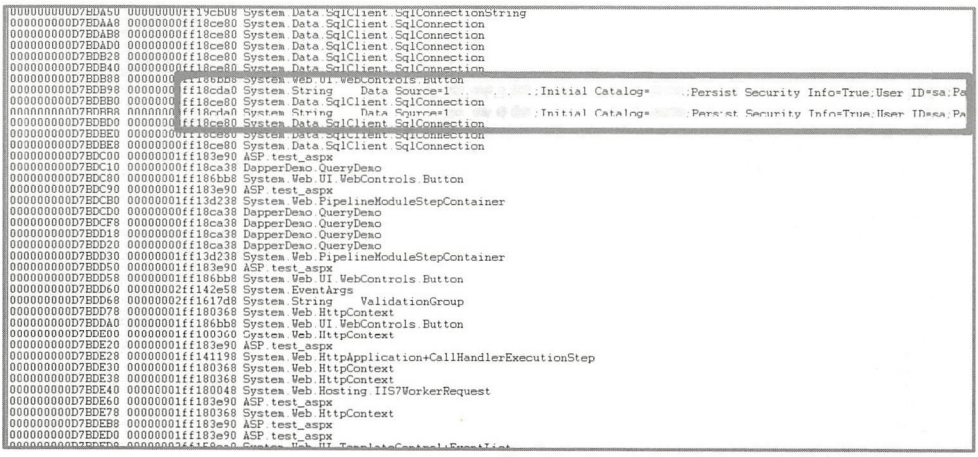
（3）使用 !dso 命令查看堆栈上的所有对象详细信息，如下图所示。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






综合以上分析，可以大胆地猜测 Common.cs 中第 16 行“Data Source=***;Initial Catalog=***;Persist Security Info=True;User ID=sa;Password=***”的这个数据库连接字符串有问题，然后到代码中相应的地方进一步确认和修改就可以了。

## 5.4 一个真实案例

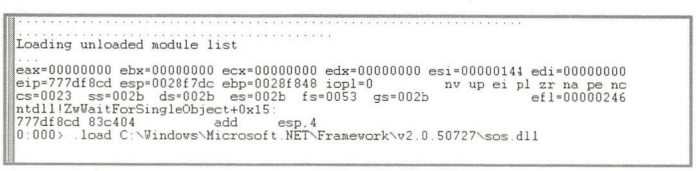
下面分享笔者工作过的一家公司某业务系统 CPU 飙高 90%以上的 Dump 分析过程案例，步骤如下。

### 1. 使用 ProcDump 抓包

### 2. 加载 SOS 扩展命令

```
.load C:\Windows\Microsoft.NET\Framework\v2.0.50727\sos.dll
```

效果如下图所示。




### 3. 分析 Dump 文件

执行“!runaway”命令，查看线程使用 CPU 时间情况，如下图所示。主要分析前面几个线程。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



```
0:000> !runaway
 User Mode Time
  Thread       Time
  22:924       0 days 0:02:46.156
  21:17d8      0 days 0:02:44.737
  17:17fc      0 days 0:02:28.700
  18:17f4      0 days 0:02:20.650
  29:ca8       0 days 0:00:26.301
  13:1408      0 days 0:00:17.409
  32:15b4      0 days 0:00:15.085
  31:9fc       0 days 0:00:14.398
  14:7e8       0 days 0:00:13.135
  33:10a8      0 days 0:00:07.987
  34:bb4       0 days 0:00:05.600
  28:fd8       0 days 0:00:05.070
  36:fcc       0 days 0:00:00.920
  37:e28       0 days 0:00:00.873
  15:1418      0 days 0:00:00.140
   6:12f8      0 days 0:00:00.140
   4:9cc       0 days 0:00:00.093
  27:9dc       0 days 0:00:00.062
   2:6f0       0 days 0:00:00.062
   0:145c      0 days 0:00:00.046
   5:240       0 days 0:00:00.031
   7:704       0 days 0:00:00.015
  35:14c8      0 days 0:00:00.000
  30:b90       0 days 0:00:00.000
  26:1658      0 days 0:00:00.000
  25:1634      0 days 0:00:00.000
  24:928       0 days 0:00:00.000
  23:14ac      0 days 0:00:00.000
  20:17dc      0 days 0:00:00.000
  19:17e0      0 days 0:00:00.000
  16:17ec      0 days 0:00:00.000
  12:764       0 days 0:00:00.000
  11:14c4      0 days 0:00:00.000
  10:1584      0 days 0:00:00.000
   9:1410      0 days 0:00:00.000
   8:1450      0 days 0:00:00.000
0:000>
```

执行“~22s”命令，进入线程 22，如下图所示。

```
   1:eb4        0 days 0:00:00.000
0:000> ~22s
eax=00000000 ebx=0fe8dafc ecx=00000000 edx=00000000 esi=00000001 edi=00000000
eip=777e0159 esp=0fe8daac ebp=0fe8db48 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
ntdll!ZwWaitForMultipleObjects+0x15:
777e0159 83c404          add     esp,4
0:022>
```

执行“!clrstack”命令查看当前线程堆栈变量值的信息，从图中可以猜出大概是ExecuteNonQuery()方法有问题，如下图所示。

```
ntdll!ZwWaitForMultipleObjects+0x15:
777e0159 83c404          add     esp,4
0:022> !clrstack
*** ERROR: Symbol file could not be found.  Defaulted to expcrt symbols for mscorwks.dll -
PDB symbol for mscorwks.dll not loaded
OS Thread Id: 0x924 (22)
ESP      EIP
0fe8dd84 777e0159 [GCFrame: 0fe8dd84]
0fe8de54 777e0159 [GCFrame: 0fe8de54]
0fe8de70 777e0159 [HelperMethodFrame_1OBJ: 0fe8de70] System.Threading.Monitor.ReliableEnter(System.Object, Boolean ByRef)
0fe8def4 713fa5d5 System.Environment+ResourceHelper.GetResourceStringCode(System.Object)
0fe8e348 71f71b4c [HelperMethodFrame_PROTECTOBJ: 0fe8e348] System.Runtime.CompilerServices.RuntimeHelpers.ExecuteCodeWithG
0fe8e3b0 713fa50f System.Environment+ResourceHelper.GetResourceString(System.String)
0fe8e3c8 713fa77b System.Environment.GetResourceStringLocal(System.String)
0fe8e60c 71f71b4c [ContextTransitionFrame: 0fe8e60c]
0fe8e5f0 71f71b4c [GCFrame: 0fe8e5f0]
0fe8e7a8 71f71b4c [HelperMethodFrame_2OBJ: 0fe8e7a8] System.Environment.GetResourceFromDefault(System.String)
0fe8e808 713b7efb System.Diagnostics.StackTrace.ToString(TraceFormat)
0fe8e874 713a9672 System.
0fe8e880 11953e37             .Service Access.DatabaseServiceBase.ExecuteNonQuery(System.Data.IDbConnection, System.Data
0fe8e964 11953d08             .Service Access.CommonDataAccess`1[[System.__Canon, mscorlib]].ExecuteNonQuery(JinRi.Domes
0fe8e97c 11953b2d             .Service Access.ProcessLogDataAccess+<>c__DisplayClass3.<Insert>b__0(System.Object)
0fe8e9c0 713da1af System.Threading._ThreadPoolWaitCallback.WaitCallback_Context(System.Object)
0fe8e9c8 71405991 System.Threading.ExecutionContext.runTryCode(System.Object)
0fe8ee54 71405887 System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext, System.Threading.Contex
0fe8ee70 713f04b5 System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext, System.Threading.ContextCallbacl
0fe8ee00 713da713 System.Threading._ThreadPoolWaitCallback.PerformWaitCallbackInternal(System.Threading._ThreadPoolWaitCal
0fe8ee9c 713da5a9 System.Threading._ThreadPoolWaitCallback.PerformWaitCallback(System.Object)
0fe8f02c 71f71b4c [GCFrame: 0fe8f02c]
0fe8f178 71f71b4c [ContextTransitionFrame: 0fe8f178]
0:022>
Ln 0, Col 0 | Sys 0:E:\work | Proc 000:15a0 | Thrd 022:924 | As
```



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



再执行“!dso”命令可以查看堆栈上的所有对象详细信息，如下图所示。


从图中可以看出，造成 CPU 飙高的原因多半由 SQL Server 执行以下这条 SQL 语句时产生异常引起，然后到源代码中找出相应的语句，经过进一步的确认、修改和重新发布后就解决了 CPU 飙高的问题。

```
INSERT INTO [dbo].[tbl_Interface_ProcessLog] (IKey,Username, ClientIP,
Module,OrderNo,LogType,Content) VALUES (@IKey,@Username,@ClientIP,@Module,
@OrderNo,@LogType,@Content)
```

掌握几个简单的 WinDbg 命令之后，基本上绝大多数 Dump 大家都可以独立分析了。当然，同时产生 CPU 飙高和内存泄漏问题的原因有很多，如果想分析得足够准确，那么就只有多学多练，多去分析。因为掌握 WinDbg 分析除了需要懂得几个命令以外，经验更加重要。最后再补充两点：

（1）WinDbg 不是专门用于调试.NET 程序的工具，它更偏向于底层，可用于内核和驱动调试，特别是对于某些疑难的问题调试有所帮助，例如内存泄漏等问题。进行普通的.NET 程序调试还是使用微软专为.NET 开发所提供的调试工具更方便一些。

（2）SOS 扩展命令中最有用的命令是“!help”，使用该命令可以列出所有可用的 SOS 扩展命令列表，使用“!help [SOSCommandName]”可以查看每一个具体扩展命令的详细使用说明。例如，“!help dumpheap”就可以查看“!dumpheap”这个扩展命令的具体使用方法。多利用“!help”命令可以很快上手 SOS。

## 5.5 Demo 下载

- WinDbgTest 下载地址：https://github.com/das2017/WinDbgTest



仅供非商业用途或交流学习使用

仅供非商业用途或交流学习使用

# 第 3 篇

# 框架篇



非卖品，仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 6 RabbitMQ 快速入门及应用

使用过分布式中间件的人都知道，中间件使用起来并不复杂，常用的客户端 API 就那么几个，比我们日常编写程序时用到的 API 要少得多。但是分布式中间件在中小研发团队中使用得并不多，为什么会这样呢？原因是中间件的职责相对单一，客户端的使用虽然简单，但整个环境搭起来却不容易。所以对于中间件的使用，我们重点放在解决门槛问题上，把服务端环境搭好（生产环境可直接使用云或运维解决），把中间件的基本职责和功能介绍好，把客户端 Demo 写好，让程序员“抬抬脚”，在调试代码中即可轻松入门。根据我们以往的经验，初次接触也可以自主快速学习。文字描述和 Demo 以实用为主，能用代码说明的就不用文字。以下是消息队列 RabbitMQ 的快速入门及应用。

## 6.1 为什么要用消息队列 RabbitMQ

### 1. 使用消息队列的好处

（1）业务系统往往要求响应能力特别强，能够起到**削峰填谷**的作用。

（2）解耦和高可用。如果一个系统“挂了”，则不会影响其他系统的继续运行。

（3）业务系统往往有对消息的高可靠要求，以及有对复杂功能（如 ACK）的要求。

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（4）增强业务系统的异步处理能力，减少甚至几乎不可能出现并发现象。

### 2. 案例说明

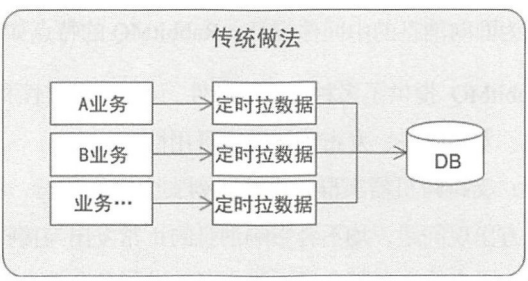
使用消息队列就好比为了防汛而建堤坝，有大量数据的堆积能力，然后可靠地进行异步输出。业务案例的传统做法存在如下问题：

（1）一旦业务处理时间超过了定时器时间间隔，就会导致漏单。

（2）如果采用新开线程的方式获取数据，那么由于处理大量新开线程，会容易造成服务器宕机。

（3）数据库压力大，易并发。

传统做法如下图所示。




基于以上问题，我们使用消息队列 RabbitMQ 改进了系统。先定时从数据库获取数据，然后存入 MQ 消息队列，最后 Job 定期扫描 MQ 消息队列进行处理，改进后有如下好处：

（1）业务可注册、可配置。

（2）数据获取规则可配置。

（3）成功消费 MQ 中的消息才会被确认（ACK），提高可靠性。

（4）大大增强了异步处理业务作业的能力，假设 Job 扫描后先预取 5 条消息，然后异步处理这 5 条消息，也就是说，这 5 条消息可能会同时被处理。

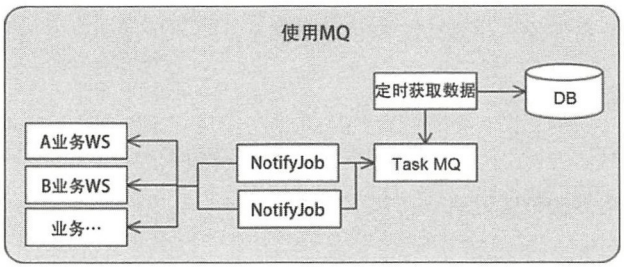
使用 MQ 的做法如下图所示。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






## 6.2 RabbitMQ 简介

RabbitMQ 是基于 AMQP 实现的一个开源消息组件，主要用于在分布式系统中存储和转发消息，由因高性能、高可用及高扩展而出名的 Erlang 语言写成。其中，AMQP（Advanced Message Queuing Protocol，即高级消息队列协议）是一个异步消息传递所使用的应用层协议规范，为面向消息的中间件设计。RabbitMQ 的特点如下。

- **高可靠**：RabbitMQ 提供了多种多样的特性，让你在可靠性和性能之间做出权衡，包括持久化、发送应答、发布确认及高可用性。
- **高可用队列**：支持跨机器集群，支持队列安全镜像备份，消息的生产者与消费者不论哪一方出现问题，均不会影响消息的正常发出与接收。
- **灵活的路由**：所有的消息都会通过路由器转发到各个消息队列中，RabbitMQ 内建了几个常用的路由器，并且可以通过路由器的组合及自定义路由器插件来完成复杂的路由功能。
- **支持多客户端**：对主流开发语言（如 Python、Ruby、C#、Java、C、PHP、ActionScript 等）都有客户端实现。
- **集群**：本地网络内的多个 Server 可以聚合在一起，共同组成一个逻辑上的 Broker。
- **扩展性**：支持负载均衡，动态增减服务器，简单方便。
- **权限管理**：灵活的用户角色权限管理，Virtual Host 是权限控制的最小粒度。
- **插件系统**：支持各种丰富的插件扩展，也支持自定义插件，其中最常用的插件是 Web 管理工具 RabbitMQ_Management，其 Web UI 访问地址为 http://139.198.13.12:6233/，登录用户名为 flight，密码为 yyabc123。



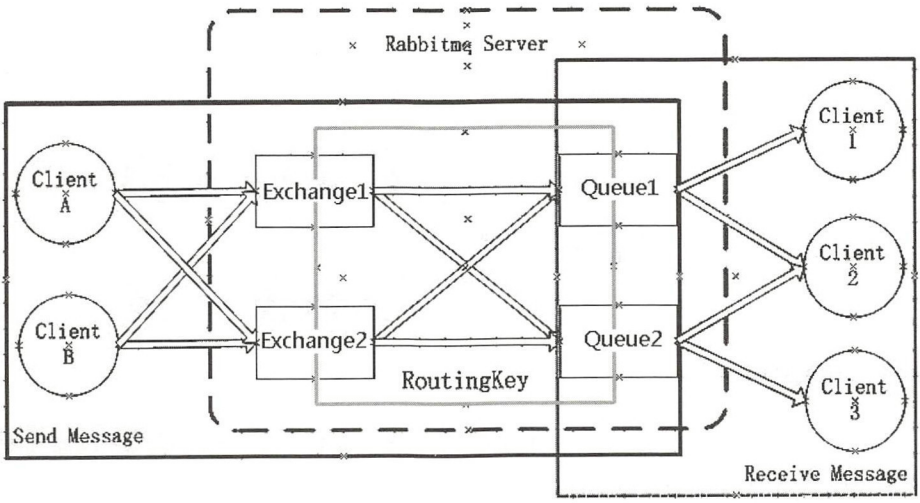
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



## 6.3 RabbitMQ 的工作原理

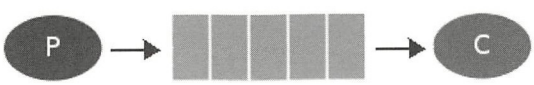
消息从发送端到接收端的流转过程，即 RabbitMQ 的消息发送与接收的工作机制，如下图所示。



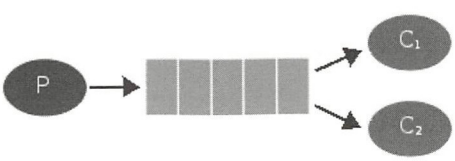

## 6.4 RabbitMQ 的基本用法

RabbitMQ 有 6 种基本用法：单对单、单对多、发布订阅模式、按路由规则发送/接收、主题和 RPC（即远程存储调用）。我们将介绍单对单、单对多和主题的用法。

- **单对单：**单发送、单接收，如下图所示。




- **单对多：**一个发送端，多个接收端，比如分布式的任务派发，如下图所示。




- **主题：**Exchange Type 为 topic。

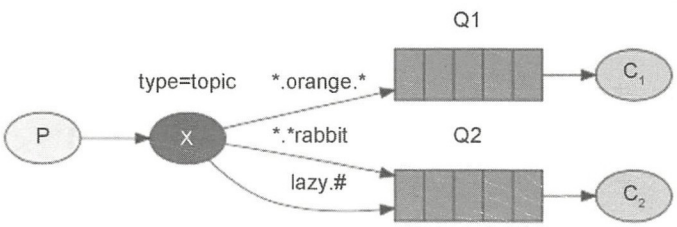


仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



发送消息时需要指定交换机及 Routing Key，消费者的消息队列绑定到该交换机并匹配到 Routing Key 实现消息的订阅，订阅后可接收消息。只有消费者将队列绑定到该交换机且指定的 Routing Key 符合匹配规则，才能收到消息。其中 Routing Key 可以设置成通配符，如*或#（*表示匹配 Routing Key 中的某个单词，#表示任意的 Routing Key 的消息都能被收到）。如果 Routing Key 由多个单词组成，则单词之间用“.”来分隔，如下图所示。




- **命名规范建议。**

交换机名的命名规范建议：Ex{AppID}.{自定义 ExchangeName}。队列名的命名规范建议：MQ{AppID}.{自定义 QueueName}。

## 6.5 Demo 下载

- RabbitMQDemo 下载地址：https://github.com/das2017/RabbitMQDemo



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 7 Redis 快速入门及应用

Redis 的使用难吗？不难。Redis 用好容易吗？不容易。Redis 的使用虽然不难，但与业务结合的应用场景特别多、特别密切，用好并不容易。我们希望通过简单的文字介绍及 Demo，读者即可轻松、快速入门并学会应用。

## 7.1 Redis 简介

Redis 是一个开源的 Key-Value 存储，但不仅仅是 Key-Value 存储，官网上的描述是，Redis 是一个数据结构存储，可用作数据库、缓存和消息中间件。相对于传统的 Key-Value 存储 Memcached，Redis 具有如下特点：

- 速度快；
- 丰富的数据结构，除 String 外，还有 List、Hash、Set、Sorted Set；
- 单线程，避免了线程切换和锁的性能消耗；
- 原子操作；
- 可持久化（RDB 与 AOF）；
- 发布/订阅；
- 支持 Lua 脚本；

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



- 分布式锁；
- 事务；
- 主从复制与高可用（Redis Sentinel）；
- 集群（3.0 版本以上）。

## 7.2 Redis 的数据结构

### 1. String

这是最简单的 Redis 类型。如果只使用这种类型，则 Redis 就像一个可持久化的 Memcached 服务器。

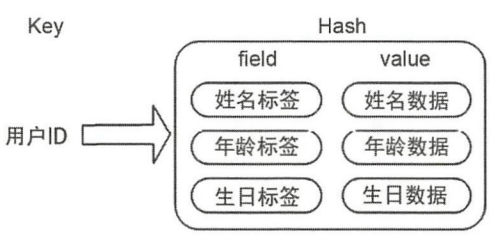
### 2. List

Redis 的 List 是基于双向链表实现的，可以支持反向查找和遍历。

常用案例：聊天系统、社交网络中获取用户最新发表的帖子、简单的消息队列、新闻的分页列表、博客的评论系统。

### 3. Hash

Hash 是一个 String 类型的 field 和 value 之间的映射表，如下图所示，类似于.NET 中的 Hashtable 和 Dictionary。主要用来存储对象，可以避免序列化的开销和并发修改控制的问题。




### 4. Set

Set 也是一个列表，不过它的特殊之处在于它是可以自动排重的：当需要存储一个列表数据，而又不希望出现重复的时候，Set 是一个很好的选择（比如 ID 的集合）。并且



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



Set 提供了判断某个成员是否在一个 Set 集合内的接口，这也是 List 所没有的。

#### 5. Sorted Set

Sorted Set 和 Set 的使用场景类似，区别是 Sorted Set 会根据提供的 score 参数来进行自动排序。当你需要一个有序的且不重复的集合列表时，那么就可以选择 Sorted Set 数据结构。常用案例：游戏中的排行榜。

## 7.3 Redis 的重要特性

下面重点介绍管道、事务和分布式锁。

#### 1. 管道

Redis 管道是指客户端可以将多个命令一次性发送到服务器，然后由服务器一次性返回所有结果。管道技术在批量执行命令的时候可以大大减少网络传输的开销，提高性能。

#### 2. 事务

Redis 事务是一组命令的集合。一个事务中的命令要么都执行，要么都不执行。如果命令在运行期间出现错误，则不会自动回滚。

管道与事务的区别：管道主要是网络上的优化，客户端缓冲一组命令，一次性发送到服务器端执行，但是并不能保证命令是在同一个事务里面执行的；而事务是原子性的，可以确保命令执行的时候不会有来自其他客户端的命令插入命令序列中。

#### 3. 分布式锁

分布式锁是控制分布式系统之间同步访问共享资源的一种方式。在分布式系统中，常常需要协调它们的动作，如果不同的系统或同一个系统的不同主机之间共享了一个/一组资源，那么访问这些资源的时候，往往需要通过互斥来防止彼此干扰以保证一致性，在这种情况下，便需要使用分布式锁。

#### 4. 地理信息

从 Redis 3.2 开始，新增了地理信息相关的命令，可以将用户给定的地理位置信息（经



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



纬度）存储起来，并对这些信息进行操作。

## 7.4 使用方法

步骤 1：在需要使用 Redis 的项目中引用 FxCommon.dll 和 Redis.dll。

步骤 2：在 App.config 或 Web.config 文件中添加如下配置。

```
    <add key="RedisServerIP" value="redis:uuid845tylabc123@139.198.13.12:
4125"/>
    <!--提供的Redis环境是单机版配置。如果Redis是主从配置，则还需设置RedisSlaveSer
verIP-->
    <!--<add key="RedisSlaveServerIP" value="redis:uuid845tylabc123@139.198.
13.13:4125"/>-->

    <!--Redis数据库。如果不需要指定Redis数据库，则配置默认值0-->
    <add key="RedisDefaultDb" value="0"/>
```

步骤 3：使用 PooledRedisClientManager 类创建 Redis 连接池。

```
    // 读取Redis主机IP配置信息
    string[] redisMasterHosts = ConfigurationManager.ConnectionStrings ["
RedisServerIP"].ConnectionString.Split(',');

    // 如果Redis服务器是主从配置，则还需要读取Redis Slave机的IP配置信息
    string[] redisSlaveHosts = null;
    var slaveConnection = ConfigurationManager.ConnectionStrings ["RedisS
laveServerIP"];
    if (slaveConnection != null && !string.IsNullOrWhiteSpace (slaveConne
ction.ConnectionString))
    {
        string redisSlaveHostConfig = slaveConnection.ConnectionString;
        redisSlaveHosts = redisSlaveHostConfig.Split(',');
    }

    // 读取RedisDefaultDb配置
    int defaultDb = 0;
    string defaultDbSetting = ConfigurationManager.AppSettings ["RedisDef
aultDb"];
    if (!string.IsNullOrWhiteSpace(defaultDbSetting))
    {
        int.TryParse(defaultDbSetting, out defaultDb);
```



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



```
    }

    var redisClientManagerConfig = new RedisClientManagerConfig
    {
        MaxReadPoolSize = 50,
        MaxWritePoolSize = 50,
        DefaultDb = defaultDb
    };

    // 创建Redis连接池
    Manager = new PooledRedisClientManager(redisMasterHosts, redisSlaveHosts,
redisClientManagerConfig)
    {
        PoolTimeout = 2000,
        ConnectTimeout = 500
    };
```

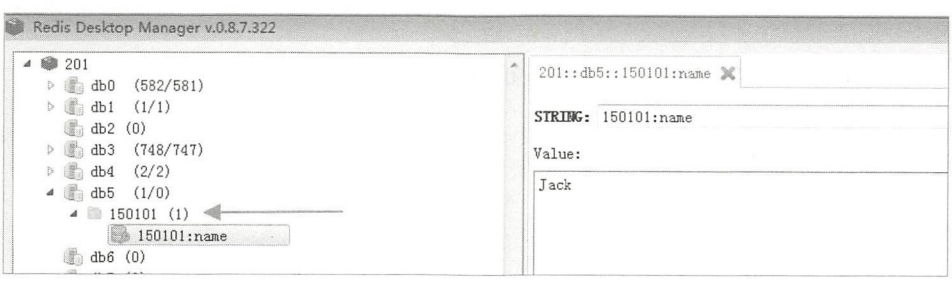
步骤 4：通过 PooledRedisClientManager 的实例获取 Redis 客户端，然后就可以通过 Redis 客户端的 API 进行操作了，详见 Demo。

## 7.5 Redis Key 命名规范与常见问题

### 1. Redis Key 命名规范

Redis Key 命名规范：AppID:KeyName。

可能有很多人习惯用英文状态的点号来作为 AppID 和 KeyName 的分隔符，笔者建议使用冒号作为 AppID 和 KeyName 的分隔符，其原因是：这么写会使 Redis Key 以 AppID 作为分类显示在 Redis Desktop Manager 中，能够快速查到要查阅的 Redis Key 对应的 Redis Value 值，如下图所示。



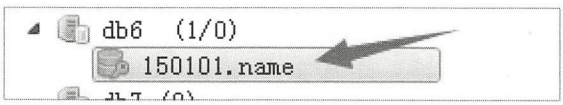



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



如果使用英文状态的点号来作为分隔符，那么在 Redis Desktop Manager 中，Redis Key 就不会被分类了，如下图所示。




### 2. 常见问题

- 缓存穿透处理：什么是缓存穿透？当根据 Redis Key 在缓存中查询后，发现不存在对应的 Value，那么会在后端系统（比如 DB）中去查找，该 Key 的并发请求量一旦变大，就会对 DB 造成很大的压力。解决办法：前端风险控制，将恶意穿透情况排除在外；对查询结果为空的情况依然进行缓存，但缓存时间会设置得很短，一般是几分钟。
- 缓存雪崩处理：什么是缓存雪崩？当缓存服务器重启或大量缓存集中在某一个时间段失效，这样在失效的时候，也会给后端系统（比如 DB）带来很大压力。解决办法：后端连接数限制，错误阈值限制，超时处理，缓存失效时间均匀分布，前端永不失效及后端主动更新。
- 缓存时长：策略定位复杂，需要多维度的计算。
- 缓存失效：按时失效、事件失效和后端主动更新。
- 缓存 Key：Hash、规则、前缀+Hash，异常情况下可人工干预。
- Lua 脚本：服务端批量处理及事务能力，有条件逻辑的可扩展脚本。使用它的好处有：减少网络开销、原子操作和可复用。
- Limit：可滑动时间窗口，比如应用于 Session，Memcached 需每次传 Key 和 Value。

## 7.6 Demo 下载

- RedisDemo 下载地址：https://github.com/das2017/RedisDemo



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 8 任务调度 Job

## 8.1 Job 简介

Job 类似于数据库中的作业，多用于实现定时执行任务。适用场景主要包括定时轮询数据库同步、定时处理数据和定时邮件通知等。我们的 Job 分为操作系统级别定时任务 WinJob 和 HttpJob，其中，WinJob 使用开源的任务调度框架 Quartz.NET+ZooKeeper 实现，HttpJob 的服务端是自主开发实现的，可以直接定时调用计划任务（如微服务）。

## 8.2 WinJob

WinJob 使用 Quartz.NET+ZooKeeper 来实现，Quartz.NET 实现调度，ZooKeeper 使用 MasterElection 来实现高可用，解决单点故障问题。ZooKeeper 后面会单独介绍，这里重点介绍 Quartz.NET 框架的使用。Quartz.NET 是一个全功能的开源任务调度框架，通过简单的配置就可以实现强大的任务调度功能，使开发人员不用过多关注任务的调度，只需关注项目的业务逻辑。

### 1. 使用任务调度框架的价值

（1）提高了开发效率：开发人员只需要编写业务代码，而具体的任务调度只需要通

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



过配置就可以实现。

（2）提高了软件的可靠性：同一应用的多个任务之间可以很好地隔离起来，互不影响。

（3）降低了开发人员成本和开发复杂度：开发人员不需要对线程和 Timer 很了解，就能实现一个强大的执行计划应用。

（4）容易迁移：只需实现 Quartz.IJob 接口，内部调用一次业务逻辑的入口即可。

（5）容易扩展：新业务只需增加配置即可。

### 2. 基于 Quartz.NET 实现 Job 调度的方法

在后端服务声明实例化一个调度器，在启动服务的时候启动调度器，相应的代码如下所示。

```
/// <summary>
/// 当前调度服务的调度器
/// </summary>
public IScheduler CurrentSched
{
   get; private set;
}
public JobService()
{
   InitializeComponent();

   StdSchedulerFactory schedulerFactory = new StdSchedulerFactory();
   CurrentSched = schedulerFactory.GetScheduler();
}
protected override void OnStart(string[] args)
{
   CurrentSched.Start();
   logger.Info("调度服务成功启动！");
}
```

创建相应的任务和触发器，之后把任务和关联的触发器加入之前声明的调度器 CurrentSched，相应的代码如下所示。

```
/// <summary>
/// 演示一个任务多触发器的使用
/// </summary>
```



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



```
    private static void JobWithManyTriggerDemo()
    {
        IJobDetail simpleJob = JobBuilder.Create<SimpleJob>().WithIdentit
y ("任务名称", "任务组名").Build(); //创建一个simpleJob任务
        ITrigger simpleTrigger = TriggerBuilder.Create().WithIdentity("触
发器名称3", "触发器组名").StartNow()
            .WithSimpleSchedule(x => x.WithIntervalInSeconds(5). RepeatFor
ever()).Build();   //创建一个简单触发器，每隔5秒执行一次

        CurrentSched.ScheduleJob(simpleJob, simpleTrigger);   //把simpleJob任
务、简单触发器加入调度器

        ITrigger cronTrigger = TriggerBuilder.Create().WithIdentity ("触发
器名称4", "触发器组名").StartNow()
            .WithCronSchedule("/10 * * ? * *").ForJob(simpleJob).Build();
    //创建一个为任务“simpleJob”服务的Cron触发器，每隔10秒执行一次

         CurrentSched.ScheduleJob(cronTrigger);   //把Cron触发器加入调度器
    }
```

在业务逻辑层继承 IJob 接口，并实现 Execute 方法，在该方法内实现需要调度的业务逻辑，相应的代码如下所示。

```
    /// <summary>
    /// 简单任务
    /// </summary>
    public class SimpleJob : IJob
    {
        ILog logger = LogManager.GetLogger(typeof(SimpleJob));
        public void Execute(IJobExecutionContext context)
        {
            Console.WriteLine("简单的任务演示！" + DateTime.Now.ToString("HH:mm:ss"));
            logger.Info("简单的任务演示！ " + DateTime.Now.ToString ("HH:mm:ss"));
            //业务逻辑处理
            Thread.Sleep(2000);
         }
    }
```

## 8.3 HttpJob

通过自主开发的 JobServer，结合自主开发的 Job 集中式管理平台，可以实现满足绝



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



大部分场景的 Job 调度。这种 Job 调度使用方式只需关注实现业务系统的业务逻辑部分即可，无须在业务系统中额外关注如何使用 Quartz.NET。

### 1. HttpJob 的服务端实现

JobServer 实现的主要逻辑：

（1）借助 Quartz，可实现多个线程（如 10 个线程）同时调用多个 HttpJob。

（2）实现了 GET、POST 和 HEAD 三种方式的请求。

（3）借助 ZooKeeper 的 MasterElection 实现高可用，实现自动主备切换。

（4）记录日志，方便追踪。

### 2. HttpJob 集中式管理平台

在集中式 Job 管理平台中配置相应的 Job 信息。配置完 Job 信息后，JobServer 获取相应的 Job 信息，就能够定时执行这些 Job。要配置的 Job 信息包括 Job 的任务名称、任务组名、请求地址、请求类型、开始时间、触发器类型、次数、间隔时间（s）、Cron-Like 表达式和状态。其中请求地址就是 JobServer 实际定时调用的任务的 HTTP 地址，例如，HttpJobDemo 的 WebForm1.aspx 任务的运行地址为 http://localhost:10786/WebForm1.aspx。

### 3. HttpJob 的优势与约束

**采用 HttpJob 的优势：**

（1）高可用——借助网站集群巧妙地解决 Job 服务的单点故障问题。

（2）方便发布——不用重启 Job 服务。

（3）减少依赖，易学易用，不用关注线程、Windows 服务方面的知识。

（4）数据分片，可以采用 URL 来取模+多个 HttpJob。

**采用 HttpJob 的约束：**

（1）由于请求 HttpJob 的最长响应时间是 30 秒，所以 Job 运行时间一旦超过 30 秒，则建议先为 Job 创建异步线程，立即返回。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（2）Job 调度的频率最少间隔时间是 1 分钟，因为通过 HttpJob 通知并不是一件高效的事情。

（3）为了安全应建立专业的 Job 集群，一般两台即可，外部不可访问，SLB 采用简单轮询方案。

（4）新增及修改 Job 配置，默认为 10 分钟生效。

## 8.4 Cron 表达式

HttpJob 调度可支持 Cron 表达式，Cron 表达式格式：[秒] [分] [小时] [日] [月] [周] [年]，具体规范如下表所示。

| 序　号 | 说　明 | 是否必填 | 允许填写的值 | 允许的通配符 |
|---|---|---|---|---|
| 1 | 秒 | 是 | 0~59 | , - * / |
| 2 | 分 | 是 | 0~59 | , - * / |
| 3 | 时 | 是 | 0~23 | , - * / |
| 4 | 日 | 是 | 1~31 | , - * ? / L W |
| 5 | 月 | 是 | 1~12 或 JAN-DEC | , - * / |
| 6 | 周 | 是 | 1~7 或 SUN-SAT | , - * ? / L # |
| 7 | 年 | 否 | empty 或 1970~2099 | , - * / |

**通配符说明：**

（1）反斜线（/）字符表示增量值。例如，在秒字段中，“5/15”代表从第 5 秒开始，每 15 秒一次。

（2）星号（*）字符是通配字符，表示该字段可以接受任何可能的值（例如，在分的字段上设置“*”，表示每一分钟都会触发）。

（3）问号（?）表示这个字段不包含具体值。如不指定日期字段，则可以在日期字段中插入“?”，表示日期值无关紧要。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（4）-表示区间，例如，在小时上设置“10-12”，表示 10、11、12 点都会触发。

（5）逗号（,）表示指定多个值，例如，在周字段上设置“MON,WED,FRI”，表示周一、周三和周五触发。

（6）井号（#）字符为给定月份指定具体的工作日。把“MON#2”放在周字段中，表示把任务安排在当月的第二个星期一。

（7）L 表示最后的意思，只用在日字段或周字段上。在日字段设置上，表示当月的最后一天。在周字段上表示星期六，相当于“7”或“SAT”。如果在“L”前加上数字，则表示该数据的最后一个。例如，在周字段上设置“6L”这样的格式，表示“本月最后一个星期五”。

（8）W 表示离指定日期的最近那个工作日（周一至周五）。例如，在日字段上设置“15W”，表示离每月 15 号最近的那个工作日触发。如果 15 号正好是周六，则找最近的周五（14 号）触发，如果 15 号是周末，则找最近的下周一（16 号）触发；如果 15 号正好在工作日（周一至周五），则就在该天触发。如果指定格式为“1W”，则表示每月 1 号往后最近的工作日触发。如果 1 号正是周六，则将在 3 号下周一触发（“W”前只能设置具体的数字，不允许区间“-”）。

（9）L 和 W 可以组合使用。如果在日字段上设置“LW”，则表示在本月的最后一个工作日触发。

**例子：**

（1）例 1：一个简单的每隔 5 分钟触发一次的表达式为“0 0/5 * * * ?”。

（2）例 2：在每隔 5 分钟的 10 秒后（例如，10:00:10 am, 10:05:10 等）触发一次的表达式为“10 0/5 * * * ?”。

（3）例 3：在每个周三和周五的 10:30、11:30、12:30 触发的表达式为“0 30 10-13 ? * WED,FRI”。

（4）例 4：在每个月的 5 号、20 号的 8 点和 10 点之间每隔半个小时触发一次且不包括 10 点，只是 8:30、9:00 和 9:30 的表达式为“0 0/30 8-9 5,20 * ?”。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



## 8.5 Demo 下载

- WinJobDemo 下载地址：https://github.com/das2017/QuartzDemo
- HttpJobDemo 下载地址：https://github.com/das2017/HttpJobDemo



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 9 应用监控系统 Metrics

## 9.1 Metrics 简介

应用监控系统 Metrics 由 Metrics.NET+InfluxDB+Grafana 组合而成，通过客户端 Metrics.NET 在业务代码中埋点，Metrics.NET 会把收集的数据存储在 InfluxDB 数据库中，然后通过 Grafana 来展示监控数据。其中，InfluxDB 服务端部署的版本号是 1.3.1，Grafana 部署的版本号是 4.0.1。下面结合这 3 个工具来介绍如何实现对应用的监控。

Metrics.NET 移植自 Java 的 metrics，它是一个给 CLR 提供度量的工具包。在业务代码中埋点 Metrics.NET 代码后，就可以方便地对各技术指标、业务指标进行度量，例如，共花多长时间完成某方法的执行、某方法在被执行的过程中共出现几次异常、某时间段内有多少订单量。Metrics.NET 提供 5 种度量类型：Gauge、Counter、Meter、Histogram 和 Timer。其中 Meter 和 Histogram 这两种度量类型目前可以完全满足笔者所在公司的度量需求，所以下面只介绍了 Meter 和 Histogram，读者若有兴趣可自行了解另外 3 个。

## 9.2 埋点 Metrics.NET 的方法

首先为需要收集 Metrics.NET 监控数据的业务项目引用 Metrics.dll。

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



然后在项目的 App.config/Web.config 文件中添加如下配置信息：

```
<add key="AppID" value="150106"/>
<add key="Metrics.DBUri" value="http://139.198.13.12:4126/write"/>
<add key="Metrics.UserName" value="Arch"/>
<add key="Metrics.Password" value="Arch"/>
<add key="Metrics.Database" value="ArchDB"/>
```

### 1. Meter

Meter 用于度量 TPS（每秒处理的请求数）。

示例：模拟统计成功下单量、下单金额和失败下单量。

调用 Meter 对象的 Mark()方法：

```
static void CreateOrder()
{
  try
  {
    // 省略关于下单的业务逻辑代码
    //......

    // 分别统计成功下单量和下单金额，统一写到MetrisKey中
    MetricsKey.OrderCount.Mark();
    if (n % 2 == 1)
    {
       MetricsKey.OrderMoneyCount.Mark("BuyerA", n);
    }
    else
    {
       MetricsKey.OrderMoneyCount.Mark("BuyerB", n);
    }
  }
  catch (Exception)
  {
    // 统计失败下单量，统一写到MetrisKey中
    MetricsKey.OrderErrorCount.Mark();

    // 省略异常处理代码
  }
}
```



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 2. Histogram

Histogram 用于度量流数据中 Value 的分布情况，它不仅能像 Meter 一样测量出 TPS，还能测量出最小值、最大值和平均值。使用场景：统计服务器的延迟时间、统计某方法共执行多长时间。

示例：模拟统计航班查询引擎方法的耗时情况。

调用 Histogram 对象的 Update()方法：

```
  private readonly Histogram searchFlightTime = MetricsHelper.Histogram
("MetricsDemo.SearchFlightTime", Unit.Custom("ms"));

  static void SearchFlight()
  {
    Stopwatch stopwatch = Stopwatch.StartNew();

    // 模拟关于航班查询的业务逻辑的代码
    Random random = new Random((int)DateTime.Now.Ticks & 0x0000FFFF);
    var n = Random.Next(100);
    Thread.Sleep(n);

    stopwatch.Stop();

    // 统计航班搜索耗时
    searchFlightTime.Update(stopwatch.ElapsedMilliseconds);
  }
```

## 9.3 Grafana 配置

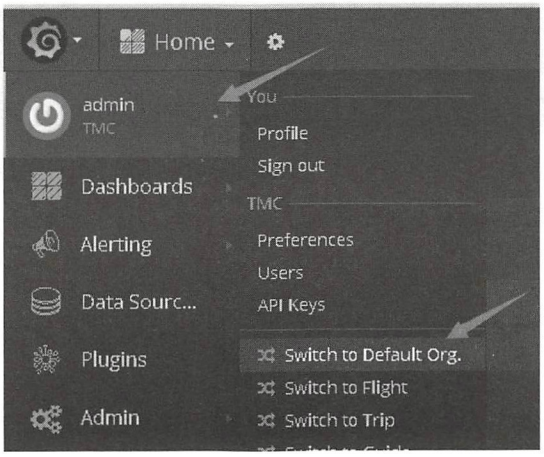
查阅 Metrics Dashboard Demo 的地址：http://139.198.13.12: 4127/。打开这个 Metrics 地址后，如果页面显示已登录状态，那么在开始查阅前，先确认是否把组织切换到了 Default Org.，如下图所示。



仅供非商业用途或交流学习使用

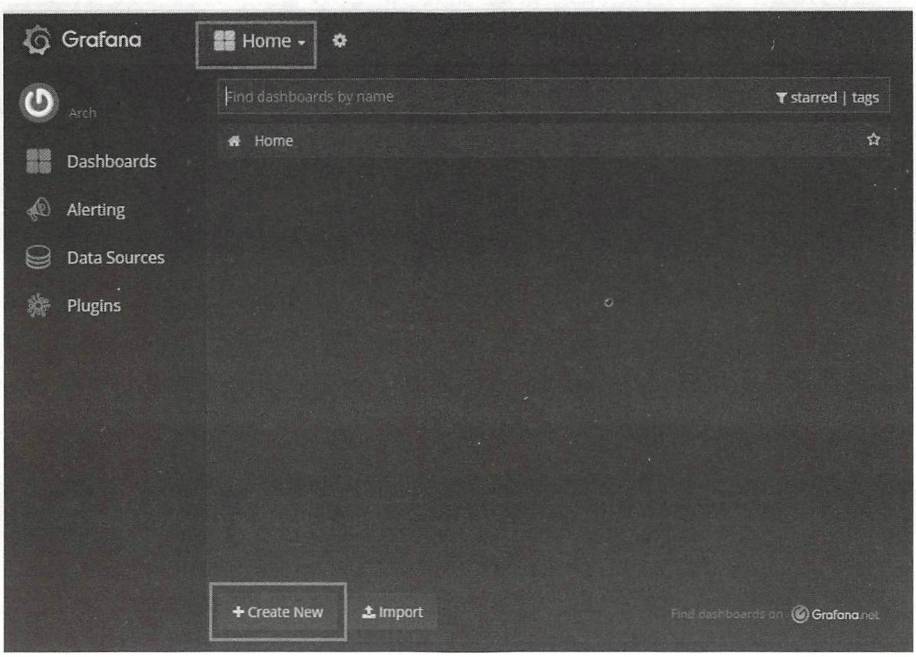
非卖品，仅供非商业用途或交流学习使用






### 9.3.1 设置仪表盘（Dashboard）

单击位于下图上方的 Home 图标，会弹出 Dashboard 列表，然后单击位于下图下方的“Create New”按钮，会进入新建面板 Panel 页面。



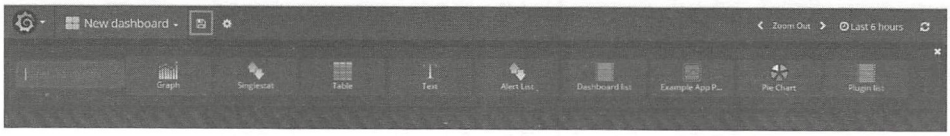

在新建面板 Panel 页面中，单击位于页面上方的保存图标按钮，如下图所示。



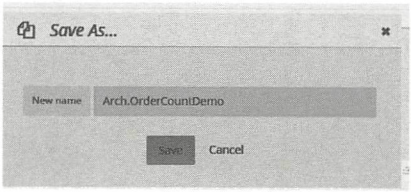
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



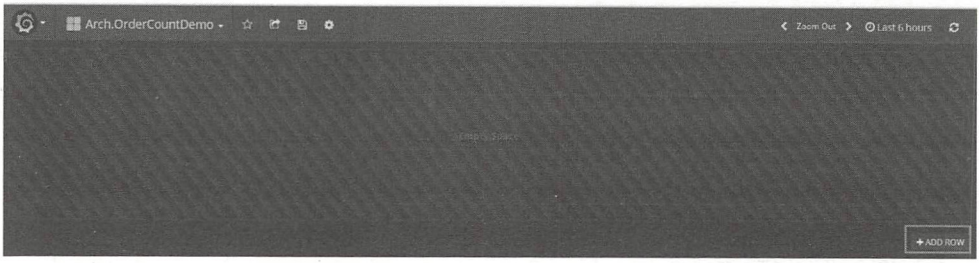



在弹出的“Save As...”对话框中输入 Dashboard 名称，如 Arch.OrderCountDemo，然后单击“Save”按钮进行保存，如下图所示。




## 9.3.2 设置面板（Panel）

单击上步创建的“Arch.OrderCountDemo”图标，进入属于该 Dashboard 的面板（Panel）页面，如下图所示。




单击上图的“ADD ROW”按钮，进入下图。其中，Graph 表示以图表（有折线图、柱状图、散点图、梯形图）形式展示数据，Singlestat 表示单个统计，Table 表示以表格形式展示数据，PieChart 表示以饼状图形式展示数据。这几种统计类型的面板设置方式类似，本节将以 Graph 为例进行说明。


### 1. 数据设置

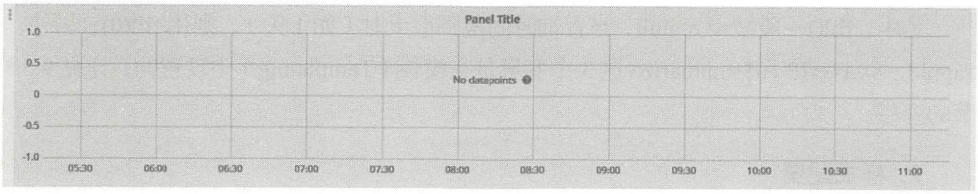
单击上图中的 Graph 图标创建图表，进入下图。



仅供非商业用途或交流学习使用

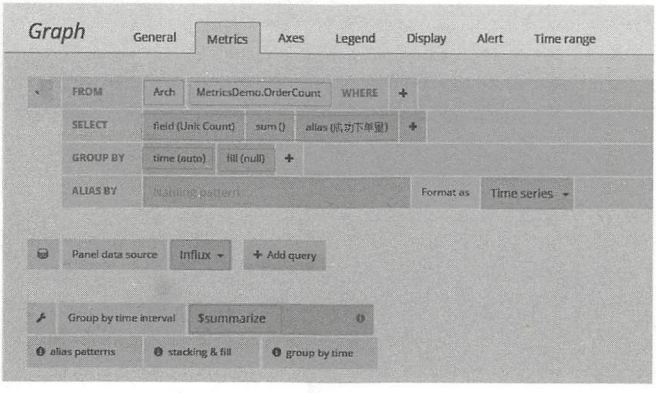
非卖品，仅供非商业用途或交流学习使用



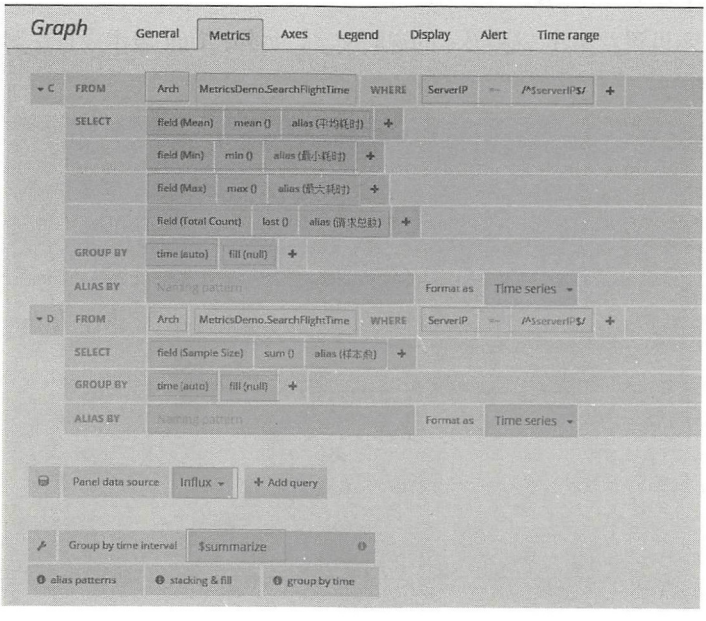



单击上图中的“Panel Title”，在弹出菜单中单击“Edit”打开 Panel 编辑界面，即进入 Metrics 选项卡面板。

Meter 的查询数据语句配置如下图所示。




Histogram 的查询数据语句配置如下图所示。






仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



其中，fill()一般被设为 null，当查询时间范围很大时（如 1 天），则用 fill(0)；另外，$appId、$serverIP 和$summarize 这 3 个变量是在模板（Templating）中设置的，详见 9.3.3 节的介绍。

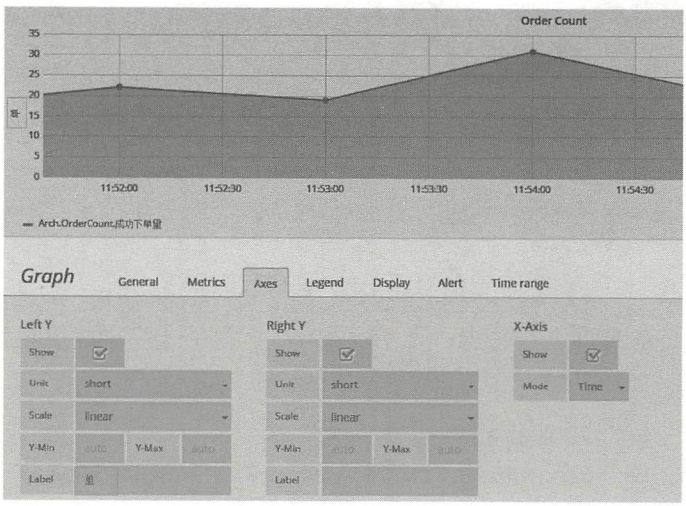
### 2. 样式配置

（1）General 选项卡用来设置 Panel 样式。

General 主要用来设置 Panel 的标题，如下图所示。


（2）Axes 选项卡用来设置坐标轴。

Label 表示设置左侧 Y 轴旁显示的说明文字，另外，Unit 表示设置左侧 Y 轴数字的单位，如下图所示。





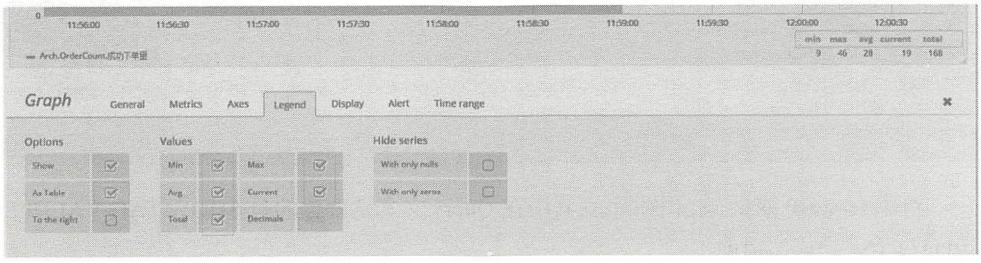

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



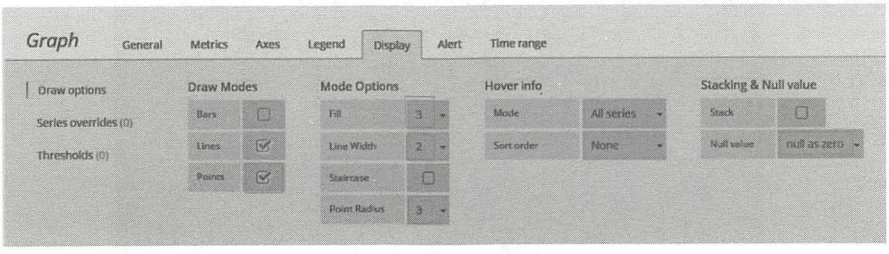
（3）Legend 选项卡用来设置显示样式。

设置内容如下图所示。




（4）Display 选项卡用来设置图表样式。

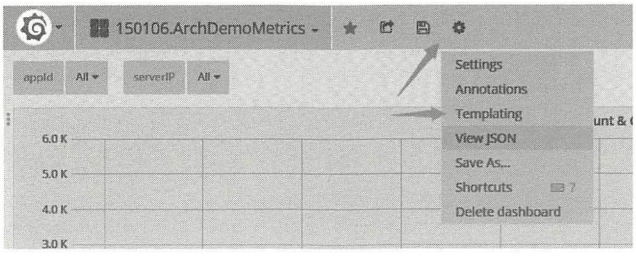
Draw options 子选项卡用来设置图表显示效果，如下图所示。




- Draw Modes：Points 表示是否在图中显示散点。
- Mode Options：Fill 表示填充度、Line Width 表示图表线的粗细、Point Radius 表示圆点半径的长度。
- Stacking&Null value：Null value 选择 null as zero 表示当该时间节点在 InfluxDB 中没有记录时，用 0 替代。

## 9.3.3 设置模板 Templating

打开 Templating 设置页面，如下图所示。



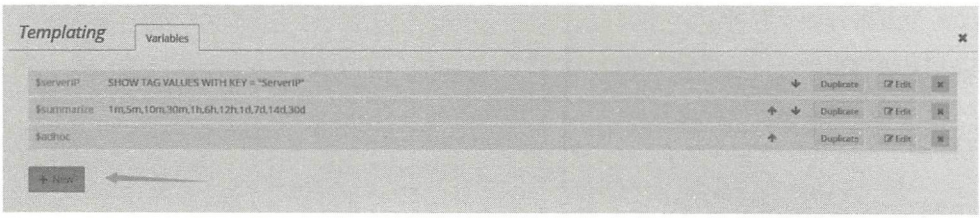



仅供非商业用途或交流学习使用

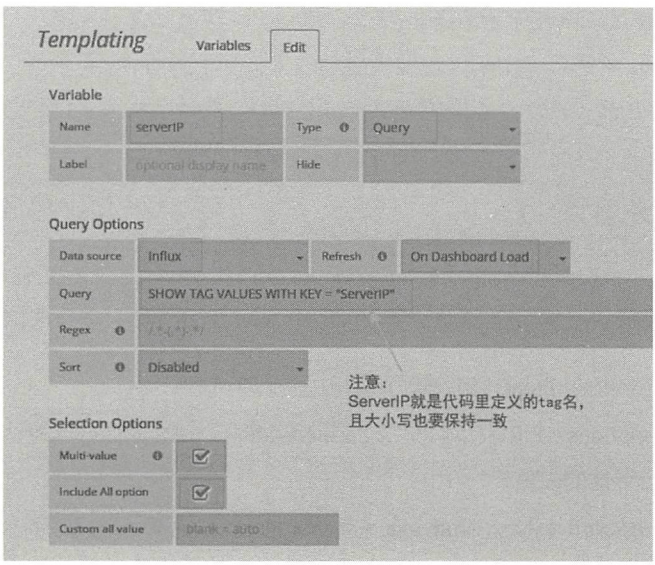
非卖品，仅供非商业用途或交流学习使用



新建变量，如下图所示。




新建 serverIP 变量，如下图所示，其中在 Query 文本框处输的是 SHOW TAG VALUES WITH KEY = "ServerIP"。



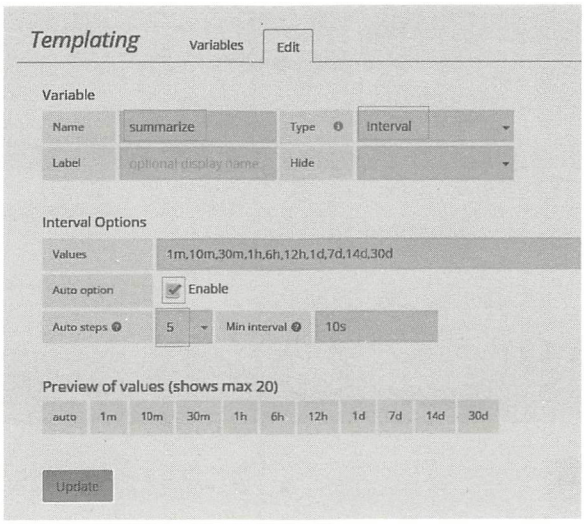

新建 summarize 变量，其中 Values 值可以自行添加或删除，值与值之间用英文状态的逗号隔开，如下图所示。



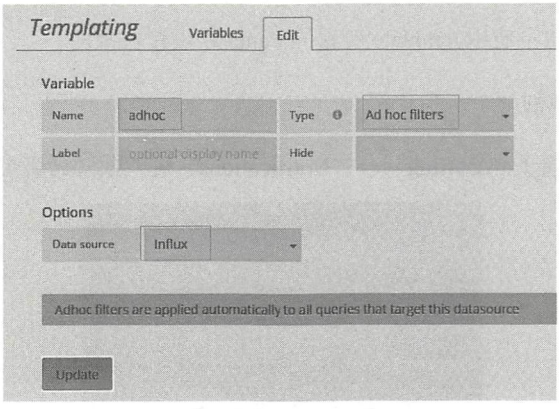
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






新建 adhoc 变量，如下图所示。




### 9.3.4 设置 Time Range

在一个 Dashboard 中，除了需要显示实时监控数据，有时还需要显示历史的监控数据，主要目的是要通过对历史监控数据的观察来预测未来的业务量走势，那么需要重写 Time Range，即需要在 Time range 选项卡中进行设置。

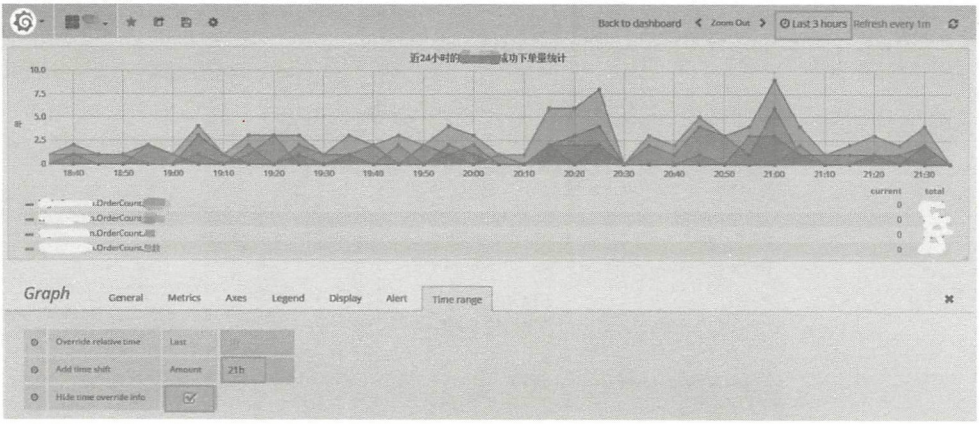
例如，在一个 Panel 中需要显示近 24 小时的历史监控数据，那么在这个 Panel 中添加配置，如下图所示。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



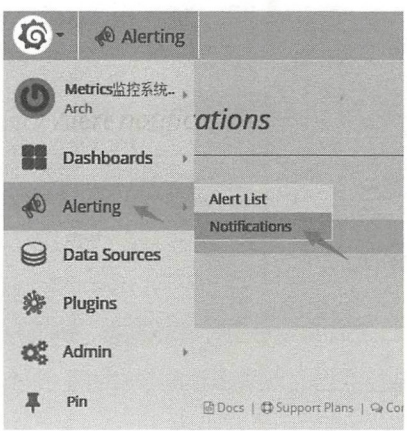



## 9.3.5 告警设置

在 Grafana 当前版本（4.0.1）中，告警目前仅支持 Graph 类型的面板，在以后的版本中会添加 Singlestat 和 Table 类型面板的支持。另外，由于告警查询语句不支持 template 变量，所以最好只对不使用 template 变量的 Panel 才设置告警。

### 1. 设置通知规则

在左侧菜单中选择“Alerting”→“Notifications”进入通知列表页，如下图所示。




单击“New Notification”按钮新建一个通知。

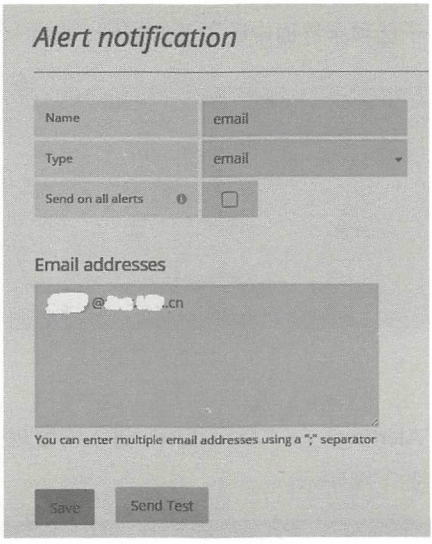
在 Name 文本输入框中，输入通知名称，类型 Type 选择 email。设置完成之后单击“Save”按钮，然后单击“Send Test”按钮测试通知是否能够发送成功，如下图所示。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

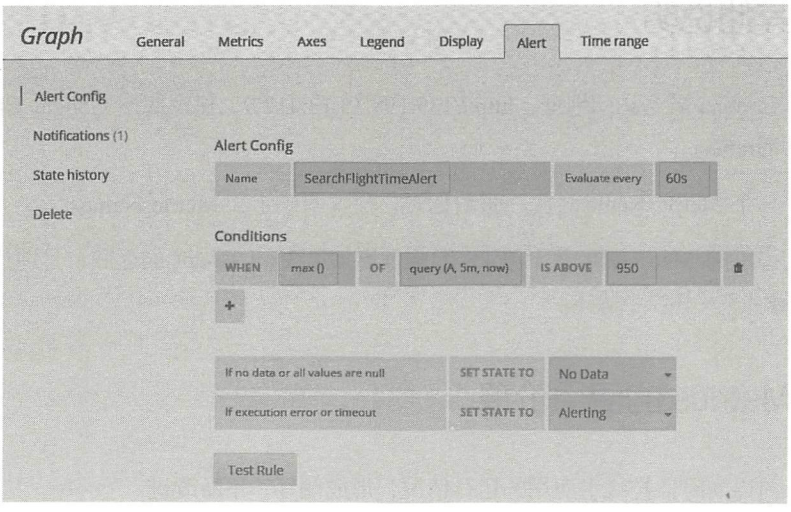





### 2. 设置告警规则

进入需要添加告警的 Panel 的编辑界面，转到 Alert 选项卡，单击“Create Alert”按钮，进入 **Alert Config 子选项卡界面**进行配置，其中 Evaluate every 表示设置执行频率，Conditions 表示配置何时告警的条件（WHEN 是选择聚合函数的地方，OF 用来设置时间段，IS ABOVE 或 IS BELOW 用来设置阈值）。

对 Alert Config 子选项卡界面的配置如下图所示。



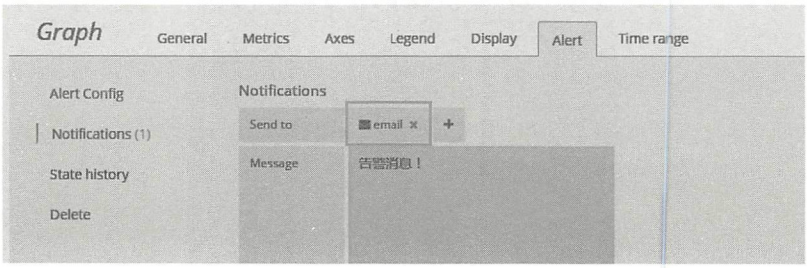



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

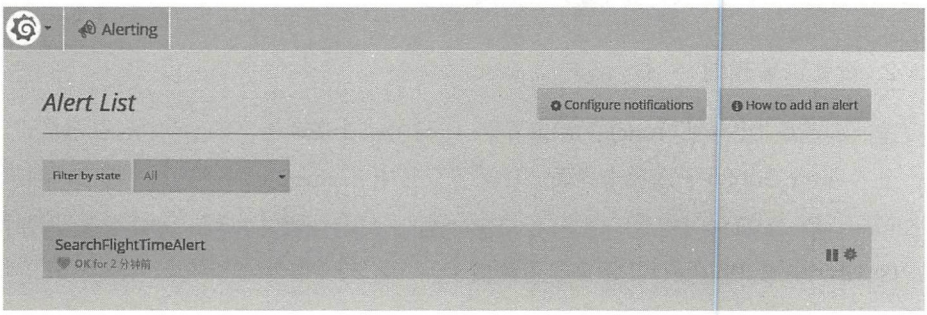


然后在 Notifications 子选项卡界面中配置通知规则，如下图所示。




### 3. 暂停告警操作

在左侧菜单中单击“Alerting”→“Alert List”进入告警规则列表页，单击暂停图标按钮就可以停止该告警，如下图所示。




## 9.4 其他说明

（1）Grafana 匿名访问地址：http://139.198.13.12:4127/。建议使用 Google Chrome 浏览器打开 Grafana。

（2）一个 MetricsName 对应一张数据表，建议明确定义 MetricsName。

（3）提供的 Metrics.dll 基于 0.4.8 的版本增加了 Unit Count 的返回，且适用于.NET Framework 4.5 及其以上版本。

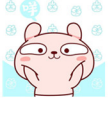
## 9.5 Metrics 的使用价值

（1）可以实时监控线上应用的运行情况，形成闭环、不断改进。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用





（2）可以预测业务未来的大致走向。

（3）可以及时发现故障，将其消灭在用户反馈之前。

（4）可设置自动报警，即时发送邮件、短信、微信（通过 API）。

（5）Metrics.NET 出现异常不影响业务流程。

## 9.6 Demo 下载

- MetricsDemo 下载地址：https://github.com/das2017/MetricsDemo



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 10 集中式日志 ELK

## 10.1 集中式日志

日志可分为系统日志、应用日志和业务日志，系统日志给运维人员使用，应用日志给研发人员使用，业务日志给业务操作人员使用。这里主要讲解应用日志，通过应用日志来了解应用的信息和状态，以及分析应用错误发生的原因等。随着系统的日益复杂，大数据时代的来临，需要几十甚至上百台的服务器是常有的事，因此迫切需要有一套针对日志且能够集中式管理的产品。ELK 就实现了集中式日志管理，统一涵盖了分布式日志收集、检索、统计、分析，以及对日志信息的 Web 管理等集中化管控。

### 1. ELK 简介

ELK 是 Elasticsearch、Logstash、Kibana 的简称，这三套开源工具组合起来能搭建一套强大的集中式日志管理平台。

- Elasticsearch 是一个开源的分布式搜索引擎，提供搜索、分析和存储数据三大功能。它的特点有：分布式、自动发现、索引自动分片、索引副本机制、RESTful 风格接口、多数据源及自动搜索负载等。
- Logstash 是一个开源的用来收集、解析和过滤日志的工具。支持几乎任何类型的

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

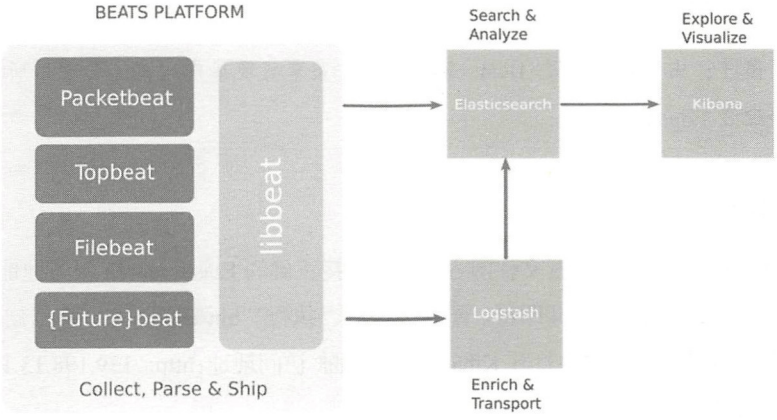


日志，包括系统日志、业务日志和安全日志。它可以从许多来源接收日志，这些来源主要包括 Syslog、消息传递（例如，RabbitMQ）和 Filebeat；能够以多种方式输出数据，这些方式主要包括电子邮件、WebSockets 和 Elasticsearch。

- Kibana 是一个基于 Web 的友好图形界面，用于搜索、分析和可视化存储在 Elasticsearch 中的数据。它利用 Elasticsearch 的 RESTful 接口来检索数据，不仅允许用户定制仪表板视图，还允许用户以特殊的方式查询、汇总和过滤数据。

### 2. ELK 的架构

下图是集中式日志管理 ELK 的架构图。出于性能的考虑，选择采用 Beats+EK 的形式来组合搭建集中式日志管理系统。




## 10.2 配置方法

### 1. Elasticsearch

Elasticsearch 部署完成后，需要更改 elasticsearch.yml 配置文件中的主要属性：cluster.name、node.name、network.host 和 discovery.zen.ping.unicast.hosts。其中，当部署 Elasticsearch 时是以集群模式部署的，那么 discovery.zen.ping.unicast.hosts 属性才会需要被配置。

### 2. Logstash

通过配置 filebeat-pipeline.conf 文件中的 Input、Filter（可选）和 Output 来完成对数据



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



的采集、过滤和输出，如下图所示。

```
input {
  beats {
    port => 5044
  }
}
output {
  elasticsearch {
    hosts => ["  .  .10.   :9200","  .  .10.   :9200"]
    manage_template => false
    index => "%{[@metadata][beat]}-%{+YYYY.MM.dd}"
    document_type => "%{[@metadata][type]}"
  }
}
```

然后以 filebeat-pipline.conf 文件启用 Logstash 服务，如下图所示。

```
    -h, --help                    print help
[root@    -vm-    logstash]# bin/logstash -f config/filebeat-pipeline.conf
Settings: Default pipeline workers: 2
Pipeline main started
```

**备注：**由于采用的是 Beats+EK 这个方案来实现集中式日志管理，所以不需要配置 Logstash。

### 3. Kibana

通过更改 kibana.yml 配置文件内容，用来连接正确的 Elasticsearch 服务地址，通常只需要配置 elasticsearch.url 属性即可。配置完成后，执行“bin/kibana &”命令启用 Kibana 服务。最后就可以在浏览器中打开 Kibana 管理页面（访问地址：http://139.198.13.12:4800/）来查看日志，如下图所示。

```
# The maximum payload size in bytes on incoming server requests.
# server.maxPayloadBytes: 1048576

# The Elasticsearch instance to use for all your queries.
elasticsearch.url: "http://localhost:9200"

# preserve_elasticsearch_host true will send the hostname specified in `elasticsearch`. If you set it to false,
# then the host you use to connect to *this* Kibana instance will be sent.
# elasticsearch.preserveHost: true
```

只需启用该属性设置

```
[root@    -vm-    kibana]# bin/kibana &
[1] 3336
[root@    -vm-    kibana]#   log   [17:39:10.195] [info][status][plugin:kibana] Status changed from uninitialized to green - Ready
  log   [17:39:10.260] [info][status][plugin:elasticsearch] Status changed from uninitialized to yellow - Waiting for Elasticsearch
  log   [17:39:10.302] [info][status][plugin:marvel] Status changed from uninitialized to yellow - Waiting for Elasticsearch
  log   [17:39:10.390] [info][status][plugin:sense] Status changed from uninitialized to green - Ready
  log   [17:39:10.403] [info][status][plugin:kbn_vislib_vis_types] Status changed from uninitialized to green - Ready
  log   [17:39:10.413] [info][status][plugin:markdown_vis] Status changed from uninitialized to green - Ready
  log   [17:39:10.493] [info][status][plugin:metric_vis] Status changed from uninitialized to green - Ready
  log   [17:39:10.693] [info][status][plugin:spyModes] Status changed from uninitialized to green - Ready
  log   [17:39:10.713] [info][status][plugin:statusPage] Status changed from uninitialized to green - Ready
  log   [17:39:10.721] [info][status][plugin:table_vis] Status changed from uninitialized to green - Ready
  log   [17:39:10.741] [info][listening] Server running at http://0.0.0.0:5601
  log   [17:39:10.757] [info][status][plugin:elasticsearch] Status changed from yellow to green - Kibana index ready
  log   [17:39:10.814] [info][status][plugin:marvel] Status changed from yellow to green - Marvel index ready
```

启用Kibana的后台服务



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 4. Filebeat

filebeat.yml 配置文件内容主要包含 Filebeat、Output、Shipper（可选）和 Logging（可选）四大部分，其中 Filebeat 主要定义监控的日志文件信息，Output 主要配置日志数据的输出目标。

filebeat.yml 文件中，主要属性值的命名规范如下：

- fields.AppID 的命名规范是{AppID}。
- fields.AppName 的命名规范是{产品线英文名称}.{项目英文名称}（如果项目英文名称由 2 个或 2 个以上英文单词组成，则单词之间用“.”分隔）。
- 针对 index 属性需要注意的是：索引（index）所定义的值是{产品线英文名称}，但英文字母必须全部小写，且不能以下画线开头，也不能包含逗号。

filebeat.yml 的配置示例如下图所示。

```
filebeat:
  prospectors:
    ################### 110107 Order Service ##########################
    -
      paths:
        - D:\Log4Net\110107\*
      fields:
        AppID: 110107
        AppName: IFlight.Order.Service
    ################### 110108 Product Service ##########################
    -
      paths:
        - D:\Log4Net\110108\*
      fields:
        AppID: 110108
        AppName: IFlight.Product.Service
output:
   elasticsearch:                    输出日志数据到elasticsearch
      hosts: ["0.  .  .  2:9200","0.  .  .  .3:9200"]
      index: logdemo
  #logstash:                         输出日志数据到logstash（已注释）
     #hosts: ["0.  .  .  1:5044"]
```

日志文件存放在哪台服务器中，filebeat 服务就部署在哪台服务器中。在 Windows 操作系统上启用 filebeat 服务的步骤：

（1）在 Windows 下开启搜索，输入 powershell，打开 powershell 所在的文件位置，右键选择 powershell.exe 以管理员身份运行，进入 PowerShell 窗口。或者以管理员身份启动 cmd.exe，输入 powershell 命令，进入 PowerShell 窗口。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



**注意：**请务必确保以管理员身份打开 PowerShell 窗口，否则在第 2 步中运行.ps1 脚本时，会报没有权限创建 filebeat 服务的错误。

（2）导向到 filebeat 执行程序所在目录。例如，cd ‘E:\ELK\filebeat-1.3.0-windows’，然后执行 powershell.exe -ExecutionPolicy UnRestricted -File .\install-service-filebeat.ps1 命令。

（3）在 PowerShell 窗口中通过以下几个命令来查看、启用和停止 filebeat 服务。

- 查看 filebeat 服务状态：Get-Service filebeat；
- 启动 filebeat 服务：Start-Service filebeat；
- 停止 filebeat 服务：Stop-Service filebeat。

## 10.3 使用方法

### 1. Log4Net 本地日志

（1）日志存放路径规范：{盘符}:\Log4Net\{AppID}\，其中 AppID 是我们所做项目的六位编码。例如，D:\Log4Net\110107\。

（2）log4net.config 的配置内容。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="log4net" type="System.Configuration.IgnoreSectionHandler"/>
  </configSections>
  <appSettings>
  </appSettings>
  <log4net>
    <appender name="FileAppender" type="log4net.Appender.RollingFileAppender">
      <!--AppID 150202，用于区分哪个应用的日志-->
      <file value="D:\Log4Net\150202\" />
      <rollingStyle value="Composite" />
      <datePattern value="yyyy-MM-dd&amp;quot;.log&amp;quot;" />
      <staticLogFileName value="false" />
      <param name="Encoding" value="utf-8" />
      <maximumFileSize value="100MB" />
```



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



```
        <countDirection value="0" />
        <maxSizeRollBackups value="100" />
        <appendToFile value="true" />
        <layout type="log4net.Layout.PatternLayout">
          <conversionPattern value="记录时间：%date 线程：[%thread] 日志级别：%
-5level 记录类：%logger 日志消息：%message%newline" />
        </layout>
      </appender>
      <logger name="FileLogger" additivity="false">
        <level value="DEBUG" />
        <appender-ref ref="FileAppender" />
      </logger>
    </log4net>
  </configuration>
```

（3）参数建议：

- maximumFileSize 设置为 100MB；countDirection 设置为大于-1 的整数；maxSizeRollBackups 设置为 100。
- 日志文件内容规范：日志文件中的每条日志内容要求是“记录时间 线程 日志级别 出错类 日志消息”。

### 2. 日志查询

基于 Kibana 查询日志（访问地址：http://139.198.13.12:4800/），主要通过以下几个步骤实现：

（1）选择业务索引库。

（2）选择日期范围。

（3）输入待查找的内容实现精确查询，也可以通过输入“*”实现模糊查询。

（4）单击每条日志的展开图标，可以查看该条日志的详细信息。

（5）在 Kibana 界面的左侧区域，自上而下依次是索引库选择框、选中字段集合列表（Selected Fields）和可用字段集合列表（Available Fields）。通过添加可用字段到选中字段集合列表中来改变 Kibana 右侧的日志表格呈现。

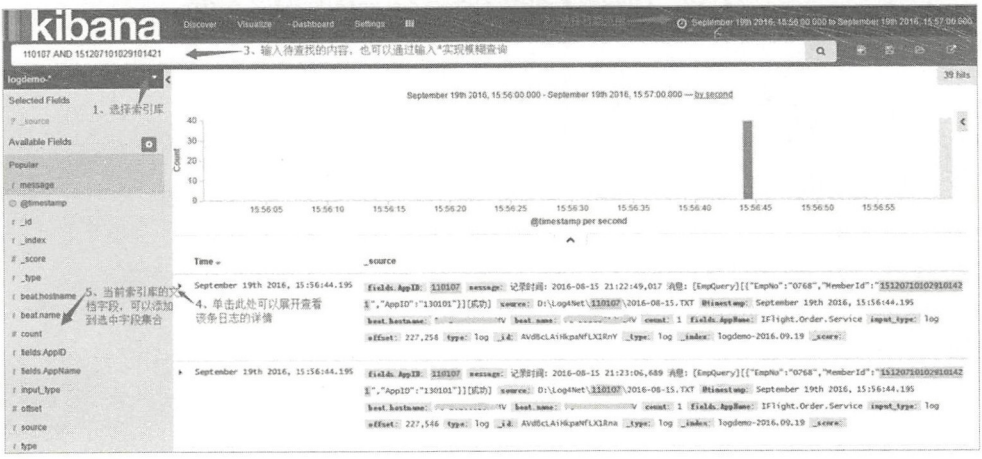


仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



Kibana 查询日志界面如下图所示。




## 10.4 Demo 下载

- Log4NetDemo 下载地址：https://github.com/das2017/Log4NetDemo



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 11 微服务架构 MSA

## 11.1 MSA 简介

### 1. MSA 是什么

微服务架构 MSA 是 Microservice Architecture 的简称，它是一种架构模式，它提倡将单一应用程序划分成一组小的服务，服务之间互相通信、互相配合，为用户提供最终价值。它与 SOA 之间的区别如下表所示。

| SOA 实现 | 微服务架构实现 |
| --- | --- |
| 企业级，自顶向下开展实施 | 团队级，自底向上开展实施 |
| 粒度大：服务由多个子系统组成 | 粒度细：一个系统被拆分成多个服务，且服务的定义更加清晰 |
| 重 ESB：企业服务总线，集中式的服务架构 | 轻网关：无集中式总线，松散的服务架构 |
| 开发过程复杂 | 易开发：减少了企业 ESB 开发的复杂性，与敏捷开发的思想高度结合在一起 |
| 单块架构系统，相互依赖，部署复杂 | 服务能独立部署 |

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 2. MSA 框架简介

本书的微服务框架 MsaFx.dll 是一个基于 ServiceStack 4.0.60 封装实现的.NET Web Services 框架，而 ServiceStack 本身支持通用的轻量级协议和 Metadata。MsaFx 与普通 Web Services 框架如 WCF 相比，主要优势如下。

（1）**高性能**：性能好、速度快。

（2）**支持跨平台运行**：基于 MsaFx 开发的 Web Services 既能够运行在 Windows 环境中，又能够运行在支持 Mono 的 Linux 环境中。

（3）**支持多协议**：如 JSON 格式的也支持 XSD。

（4）**更加 Web 化**：支持 RESTful。

（5）**服务端实现与客户端实现完全解耦**：MSA 基于消息的设计，使服务端的 API 改变不会破坏现有的客户端，达到服务端实现与客户端实现完全解耦的目的。

（6）**MSA API 可视化说明文档便于调试。**

（7）**易学**：使用 MSA 进行开发和维护服务所需的技术和时间投入要小得多。

（8）**易用**：简化了 REST 和 WCF SOAP 风格的 Web Services 的开发过程。

### 3. MSA 框架实现架构

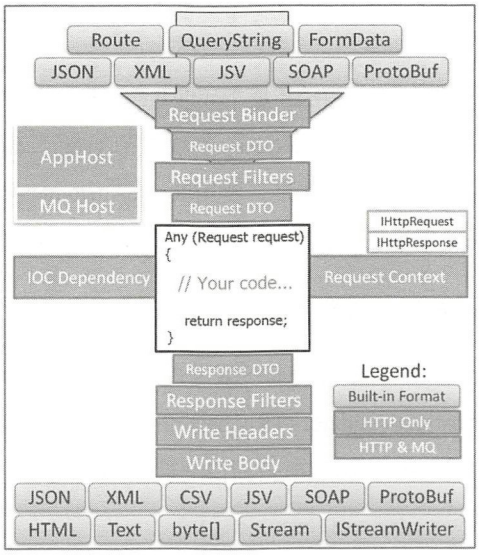
MSA 框架的服务端架构见下面的第一张图，MSA 框架的 HTTP 客户端架构见下面的第二张图。MSA 的内部是建立在原生的 ASP.NET IHttpHandler 之上实现的，支持 JSON、XML、JSV、HTML、Message Pack、ProtoBuf 和 CSV 等消息格式。



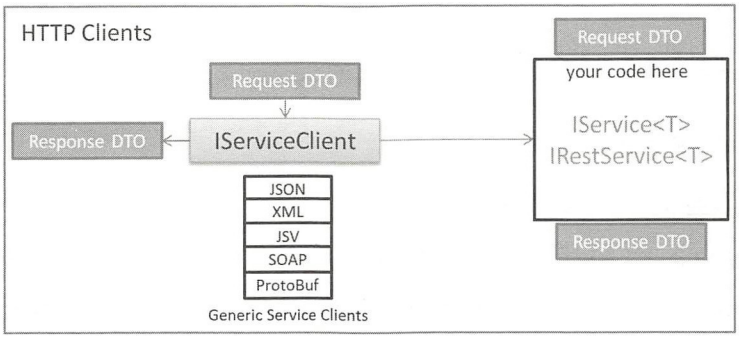
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






MsaFx 服务端架构




MsaFx HTTP 客户端架构

## 11.2 MSA 框架的使用

### 1. 服务托管

服务端对外提供服务前，必须先把服务端给托管起来。MsaFx 通过 IIS、Self-Host 等多种形式把服务端给托管起来，宿主环境可以是控制台应用、Windows Service、ASP.NET Web 应用或 ASP.NET MVC 应用。提供的 MSA Demo 的宿主环境用的是 ASP.NET Web 应用。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 2. 路由

- MsaFx 自身提供的默认路由是/[xml|json|html|jsv|csv]/[reply|oneway]/[Request DTO 名] [(?query 参数 1={值}&query 参数 2={值}&.....&query 参数 n={值})]。
- 创建自定义路由，其创建方法是：使用 RouteAttribute 或在宿主环境中配置。提供的 MSA Demo 采用的是在宿主环境中配置路由这种方式来创建自定义路由。

### 3. 如何验证请求参数的合法性

如果需要在提交请求参数前验证请求参数是否必填或是否合法，那么验证逻辑必须写在继承自 MsaFx 的 AbstractValidator<TRequest>的类中（参考例子请见 MSA Demo 的 OrderValidator.cs），然后在宿主环境中进行开启验证的配置：

```
Plugins.Add(new ValidationFeature());
container.RegisterValidator(typeof(OrderValidator));
```

### 4. 服务

创建 MSA 服务时，必须继承来自 MsaFx 的 Service 类。

### 5. MsaFx 内置的客户端

（1）MsaFx 内置了一些便捷访问的客户端，这些对象都实现了 IServiceClient 接口，其中支持 REST 的客户端都实现了 IRestClient 接口。这些客户端对象包括 JsonServiceClient、JsvServiceClient、XmlServiceClient、MsgPackServiceClient、ProtoBufServiceClient、Soap11ServiceClient 和 Soap12ServiceClient 等。从名称可以看出，这几种客户端的不同之处在于支持的序列化和反序列化格式不同。因为它们实现的是相同的接口，所以它们的用法相同，也可以相互替换。

MSA Demo 中用到了 JsonServiceClient 和 ProtoBufServiceClient 这两种客户端，其中当用到 ProtoBufServiceClient 客户端时，还需要完成如下工作。

① 除了需要引用 MsaFx.dll，还需要引用 protobuf-net.dll。

② 需要在宿主环境中进行如下配置：

```
Plugins.Add(new ProtoBufFormat());
```



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



③ 必须分别给 Request DTO 对象和 Response DTO 对象的各个属性标上[DataMember(Order = {0})]特性，具体写法请见 MSA Demo 的 ProductRequestDTO.cs 和 ProductResponseDTO.cs。

（2）MsaFx 内置的客户端提供 Get、Send、Post、Put 和 Delete 等方法。查询数据一般用 Get 方法，新增操作一般用 Post 方法，更新操作一般用 Put 方法，删除操作一般用 Delete 方法。这些方法都有重载。以下是 Get 方法的其中一个签名：

```
TResponse Get<TResponse>(IReturn<TResponse> requestDto);
```

### 6. MSA API 可视化说明文档自动生成的实现

在宿主环境中添加如下配置：

```
Plugins.Add(new SwaggerFeature());
```

如果需要在 MSA API 可视化说明文档中看到各个请求参数、响应的含义说明，那么需要为 Request DTO 和 Response DTO 对象的各个属性标上 ApiMember，代码参考如下：

```
public class OrderRequest : IReturn<OrderResponse>
{
    [ApiMember(Name = "Id", Description = "订单ID号", IsRequired = false)]
    public int Id { get; set; }
    [ApiMember(Name = "CustomerName", Description = "客户名", IsRequire
d = false)]
    public string CustomerName { get; set; }
    //......
    [ApiMember(Name = "OrderItemList", Description = "订购的产品列表", Is
Required = false)]
    public List<OrderItem> OrderItemList { get; set; }
}
```

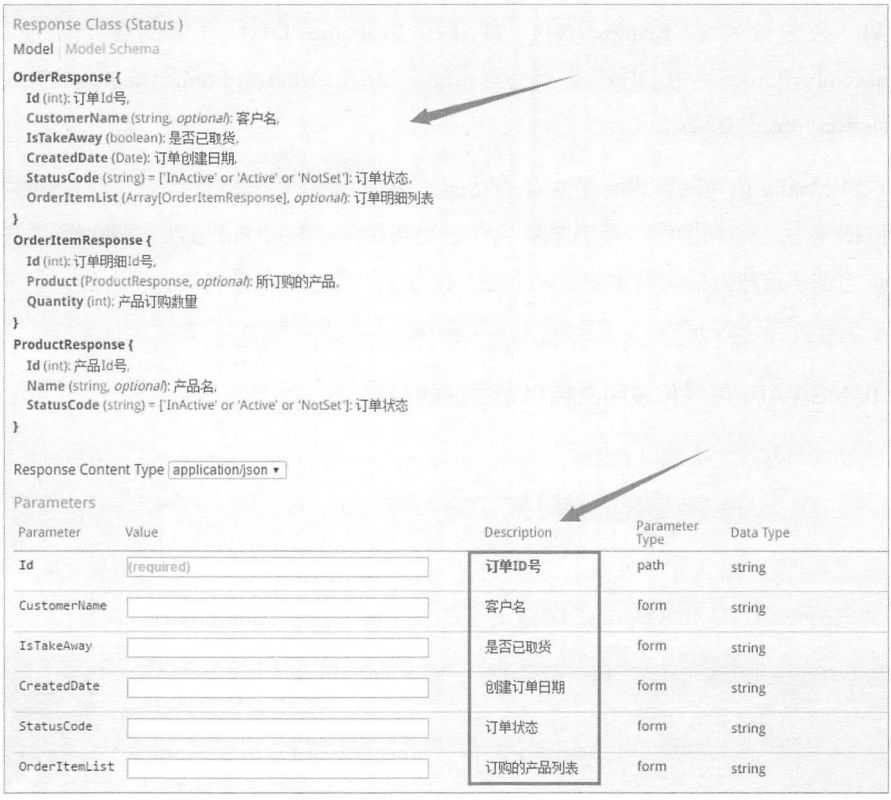
运行结果如下图所示。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






### 7. 运行结果

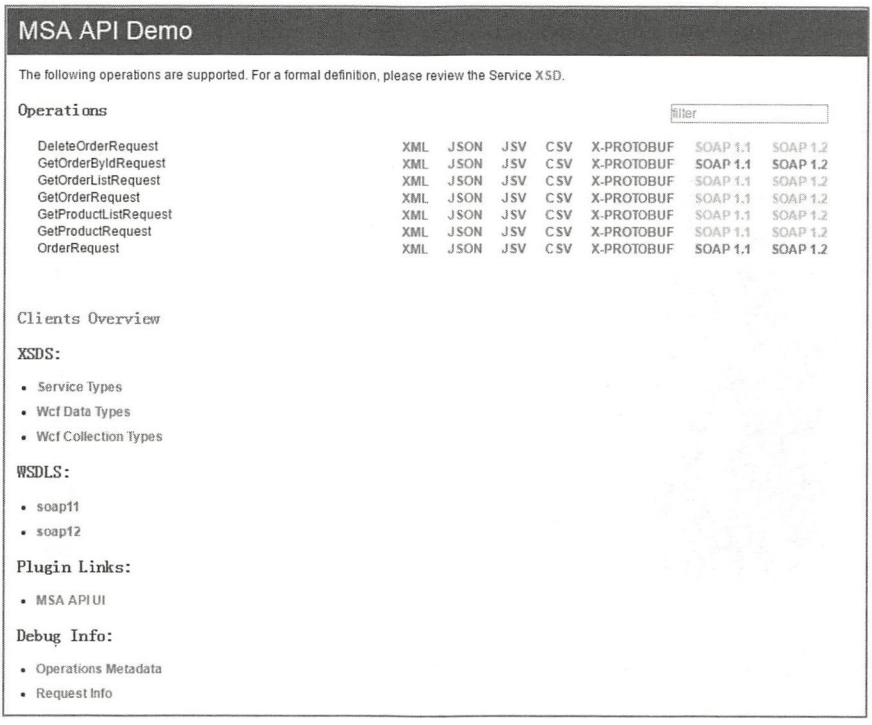
先运行托管应用（如 MSA Demo 中 ServiceHost 项目），出现下图所示的 Metadata 页。然后运行客户端（如 MSA Demo 中 Client 项目）来调用微服务；也可通过浏览器查看数据，网址输入格式为 http://localhost:34833/orders/1.html?CustomerName=客户_1&IsTakeAway=true&StatusCode=1&CreatedDate=2017-08-21，或 http://localhost:34833/tml/reply/GetOrderRequest?Id=1&CustomerName=客户_1&IsTakeAway=true&StatusCode=&CreatedDate=2017-08-21。其中，第 1 个网址格式规则就是 MSA Demo 在宿主环境中所配置的自定义路由规则，第 2 个网址格式规则就是由 MsaFx 提供的默认路由规则。



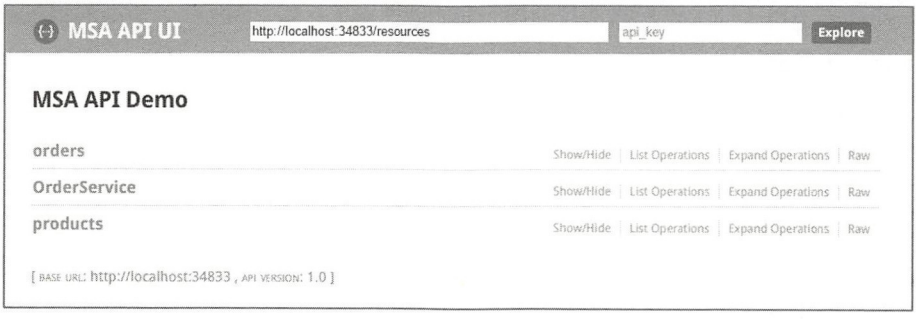
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






单击 Metadata 页中的“MSA API UI”后，进入 MSA API 可视化说明文档界面，如下图所示。开发人员可以通过这份由 MsaFx 自动生成的说明文档进行调试，十分方便。




## 11.3 微服务治理

在我们自主开发的框架管理系统中进行接口注册，如下图所示。其中，规定内部服务访问名的命名规范是“/{***Service}/方法名”，如/OrderService/CreateOrder；规定外部

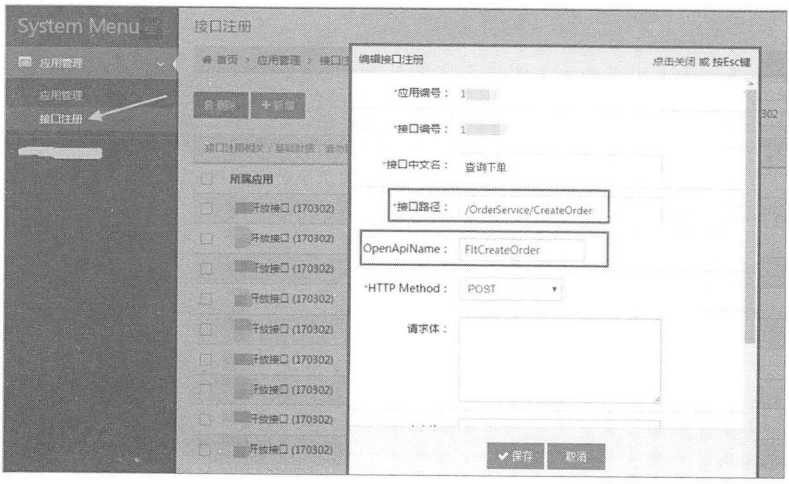


仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



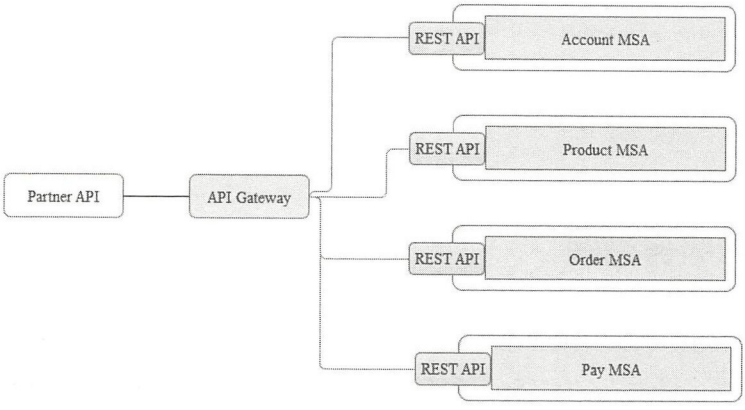
服务访问名 OpenApiName 的命名规范是“{各产品线的缩写英文名}方法名”，如 FltCreateOrder，其中 Flt 表示国内机票业务的缩写英文名。




## 11.4 微服务网关 API Gateway

### 1. API Gateway 简介

API Gateway 风格的核心理念是使用一个轻量级的消息网关作为所有客户端的主入口，并且在 API Gateway 层面实现通用的非功能性需求。如下图所示，所有的服务通过 API 网关来暴露，这是所有客户端访问的唯一入口；一个服务要访问另一个服务，也要通过这个网关。






仅供非商业用途或交流学习使用

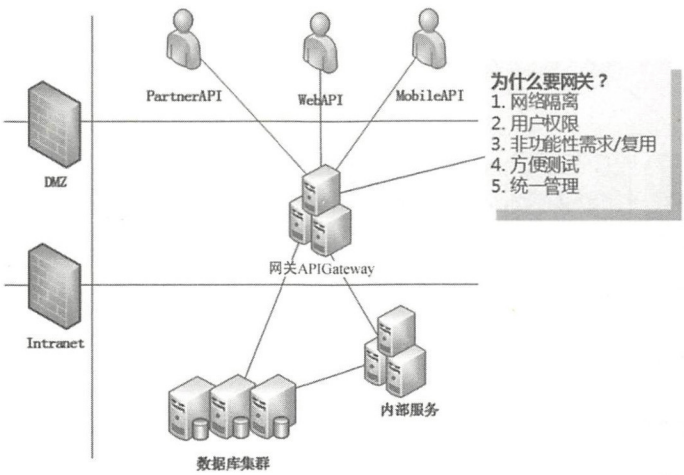
非卖品，仅供非商业用途或交流学习使用



一旦 API 网关允许客户端消费一个受管理的 API，那么我们就可以以受管理的 API 形式使用它来暴露这个微服务所实现的业务逻辑。API 网关以 NIO、IOCP 来连接内部受管理的 API，以实现 API 网关的高并发。

### 2. API Gateway 的优点

API Gateway 的层次如下图所示。




- 网络隔离：微服务部署在了内网，通过 API Gateway 开放给 PartnerAPI、WebAPI 或 MobileAPI。
- 在网关层面的轻量级消息路由和转换。
- 在网关层面对存在的微服务提供必要的抽象。例如，网关可以选择对不同的用户暴露不同的 API。
- 一个中心的地方提供非功能性的能力，这些能力可复用，比如超时、限流、熔断、监控和日志记录等。
- 通过使用 API 网关模式，微服务可以变得更加轻量，因为非功能性需求都在网关上实现了。
- 统一安全管控。

### 3. API Gateway 的架构

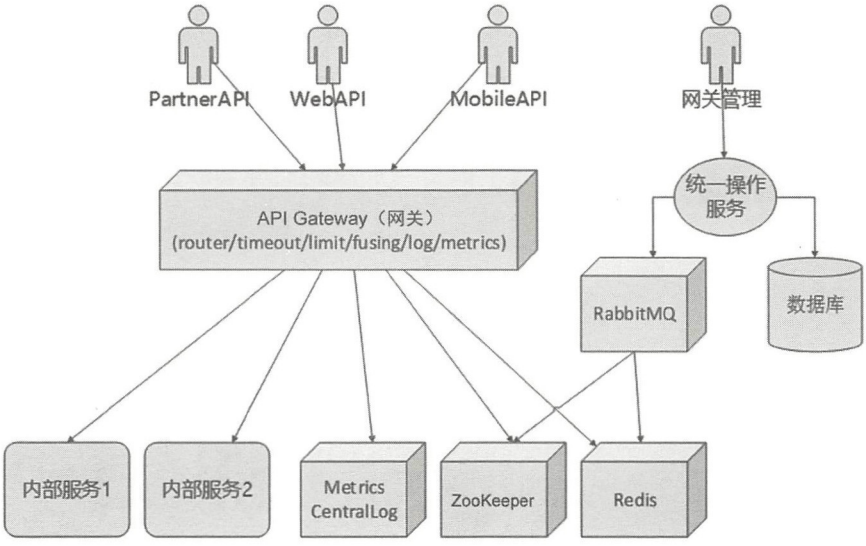
API Gateway 的架构如下图所示。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






### 4. API Gateway 的功能

API Gateway 主要实现以下功能。

（1）**路由映射**：外部服务访问名映射到对应的内部服务访问名。

（2）**权限验证**：包括针对客户角色的访问授权验证、针对客户的访问授权验证、IP 黑名单验证。

（3）**超时处理**：当 API 网关调用的内部服务响应时间超过了在自主开发的 API 网关后台管理子系统中所设置的允许最长的超时时间时，API 网关会立即停止调用，并返回相关消息给客户端。

（4）**限流控制**：当客户端通过 API 网关调用内部服务的频率达到某个阈值时，API 网关会立即做断开链路处理。过了时间后，链路会自动闭合回去。

（5）**熔断处理**：熔断处理对避免无谓的资源消耗特别有用，当通过 API 网关调用的内部服务出现异常的频率达到某个阈值时，那么 API 网关会做临时熔断处理即临时断开链路，暂时停止客户端对那个内部服务的调用。临时熔断后，再过一段时间，链路会自动闭合回去。

（6）**日志信息记录**：记录客户 IP、客户请求参数、返回结果和异常信息等信息。



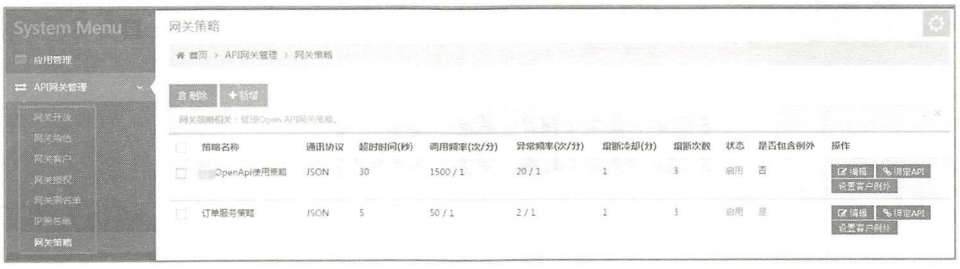
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 5. API Gateway 的使用

在使用 API Gateway 之前，需要先配置网关参数。网关参数的配置是在自主开发的 API 网关后台管理子系统中进行的，如下图所示。



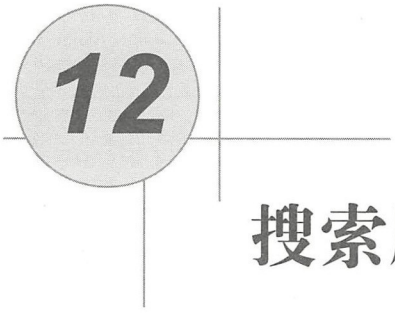

## 11.5 Demo 下载

- MSADemo 下载地址：https://github.com/das2017/MSADemo
- APIGatewayDemo 下载地址：https://github.com/das2017/ApiGatewayDemo



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



# 搜索服务 Solr

## 12.1 Solr 简介

### 1. 为什么要用搜索服务

- 模糊查询：在公司后台历史订单查询的应用中，模糊查询的实现方式为 LIKE%something%，性能很差。
- 海量数据：在业务数据库大量日志记录中，基于关键字的内容需要快速检索。
- 分库分表：分库分表后的关联查询的优化方案。

### 2. Solr 是什么

Apache Solr 是一个开源的搜索服务器，Solr 使用 Java 语言开发，主要基于 HTTP 和 Apache Lucene 实现。Apache Lucene 是一个高效的、基于 Java 的全文检索库。另外一个基于 Lucene 的搜索服务器是 Elasticsearch，由于项目历史原因，以及工程师有 Solr 的使用经验，我们选择了 Solr 而不是 Elasticsearch。如果是一个全新的项目，则 Elasticsearch 也是当下不错的选择。

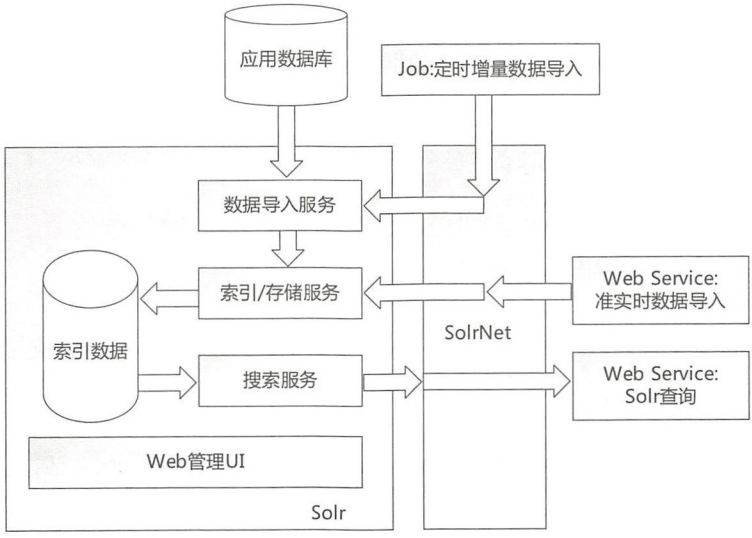
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



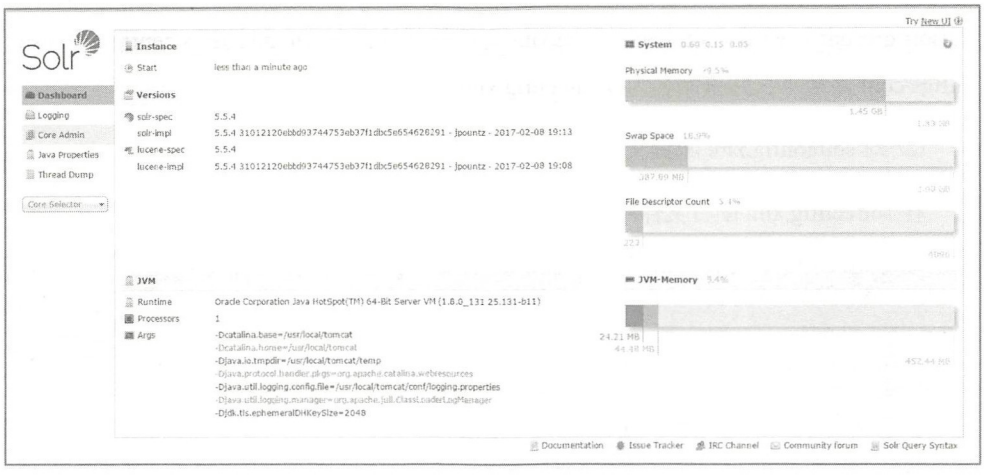
## 12.2 Solr 的工作原理

Solr 的工作原理如下图所示。




### 1. Web 管理 UI

Web 管理页面如下图所示，其 URL 为 http://139.198.13.12:7000/solr/admin.html。注意：Solr 5.5 一定要加 admin.html，如果不加，则按回车后将返回 404（表示找不到页面）。






仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



## 2. Solr 服务端的安装与配置

（1）安装 Solr 服务：安装的版本号是 5.5.4。

（2）建立 Core。

要想使用 Solr，需要先建立类似于数据库实例的 Core。每个 Core 对应一个文件夹，此文件夹建立在 Solr Home 路径下，且名字要和 Core 的名字一致，如下图所示。

```
[root@    01 solrhome]# pwd
/usr/local/solrhome
[root@j   01 solrhome]# ls
configsets  contrib  dist  OrderCore  PolicyCore  README.txt  solr.xml  zoo.cfg
```

（3）配置 Core。

以 Demo 中使用 Solr 服务器上的 PolicyCore 为例，修改下图中的 3 份配置文件。

```
[root@    01 conf]# pwd
/usr/local/solrhome/PolicyCore/conf
[root@    01 conf]# ls
admin-extra.html            currency.xml          lang              rest_managed.json  synonyms.txt
admin-extra.menu-bottom.html data-config.xml      managed-schema    solrconfig.xml
admin-extra.menu-top.html   dataimport.properties protwords.txt     stopwords.txt
```

① 3 份配置文件的来源。

solrconfig.xml、managed-schema 从位于{Solr Home 路径}/configsets/basic_configs/conf 路径下的同名配置文件复制而来。而 data-config.xml 的来源：对 Solr 服务端安装文件 solr-5.5.4.tgz 解压后，得到 solr-5.5.4 的文件夹名，然后把位于 solr-5.5.4/example/example-IH/solr/db/conf 路径下的 db-data-config.xml 文件复制到{Solr Home 路径}/configsets/basic_onfigs/conf 路径下，并重命名为 data-config.xml。

② 对 solrconfig.xml 配置文件进行修改。

在 solrconfig.xml 配置文件中增加如下内容：

```
<lib dir="../contrib/extraction/lib" regex=".*\.jar" />
<lib dir="../dist/" regex="solr-cell-\d.*\.jar" />
<lib dir="../contrib/clustering/lib/" regex=".*\.jar" />
<lib dir="../dist/" regex="solr-clustering-\d.*\.jar" />
<lib dir="../contrib/langid/lib/" regex=".*\.jar" />
<lib dir="../dist/" regex="solr-langid-\d.*\.jar" />
<lib dir="../contrib/velocity/lib" regex=".*\.jar" />
<lib dir="../dist/" regex="solr-velocity-\d.*\.jar" />
```



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



```
<lib dir="../dist/" regex="solr-dataimporthandler-\d.*\.jar" />
```

以上内容加在<luceneMatchVersion>5.5.4</luceneMatchVersion>节点之后、<dataDir>${solr.data.dir:}</dataDir>节点之前。

然后在此配置文件中增加如下内容：

```
<requestHandler name="/dataimport" class="solr.DataImportHandler">
    <lst name="defaults">
        <str name="config">data-config.xml</str>
    </lst>
</requestHandler>
```

以上内容加的位置如下图所示。

```
<!-- A request handler for demonstrating the terms component -->
<requestHandler name="/terms" class="solr.SearchHandler" startup="lazy">
   <lst name="defaults">
    <bool name="terms">true</bool>
    <bool name="distrib">false</bool>
  </lst>
  <arr name="components">
    <str>terms</str>
  </arr>
</requestHandler>

<requestHandler name="/dataimport" class="solr.DataImportHandler">
    <lst name="defaults">
        <str name="config">data-config.xml</str>
    </lst>
</requestHandler>

<!-- Legacy config for the admin interface -->
<admin>
  <defaultQuery>*:*</defaultQuery>
</admin>
```

③ 对 managed-schema 配置文件进行修改。

以下内容加在<schema>节点内：

```
<fieldType name="textPolicy_ik" class="solr.TextField">
    <analyzer type="index" useSmart="false" class="org.wltea.analyzer.
lucene.IKAnalyzer" />
    <analyzer type="query" useSmart="true" class="org.wltea.analyzer.
 lucene.IKAnalyzer" />
</fieldType>
```

然后注释掉以下配置：

```
<field name="id" type="string" indexed="true" stored="true" "required
= "true" multiValued="false" />
```



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



再在其下增加如下配置：

```
   <field name="PolicyID" type="string" indexed="true" stored="true" req
uired="true" multiValued="false" />
   <field name="PolicyGroupID" type="long" indexed="true" stored="true" />
   <field name="PolicyOperatorID" type="long" indexed="true" stored="true" />
   <field name="PolicyOperatorName" type="textPolicy_ik" indexed="true"
stored="true" omitNorms="true" />
   <field name="PolicyCode" type="textPolicy_ik" indexed="true" stored=
"true" omitNorms="true" />
   <field name="PolicyName" type="textPolicy_ik" indexed="true" stored=
"true" omitNorms="true" />
   <field name="PolicyType" type="string" indexed="true" stored="true" />
   <field name="TicketType" type="int" indexed="true" stored="true" />
   <field name="FlightType" type="int" indexed="true" stored="true" />
   <field name="DepartureDate" type="tdate" indexed="true" stored="true"
 default="NOW+8HOUR" />
   <field name="ArrivalDate" type="tdate" indexed="true" stored="true" d
efault="NOW+8HOUR" />
   <field name="ReturnDepartureDate" type="tdate" indexed="true" stored=
 "true" default="NOW+8HOUR" />
   <field name="ReturnArrivalDate" type="tdate" indexed="true" stored= "
true" default="NOW+8HOUR" />
   <field name="DepartureCityCodes" type="textPolicy_ik" indexed="true"
stored="true" omitNorms="true" />
   <field name="TransitCityCodes" type="textPolicy_ik" indexed="true" st
ored="true" omitNorms="true" />
   <field name="ArrivalCityCodes" type="textPolicy_ik" indexed="true" st
ored="true" omitNorms="true" />
   <field name="OutTicketType" type="int" indexed="true" stored="true" />
   <field name="OutTicketStart" type="tdate" indexed="true" stored="true
" default="NOW+8HOUR" />
   <field name="OutTicketEnd" type="tdate" indexed="true" stored="true"
default="NOW+8HOUR" />
   <field name="OutTicketPreDays" type="int" indexed="true" stored="true" />
   <field name="Remark" type="textPolicy_ik" indexed="true" stored="true
" omitNorms="true" />
   <field name="Status" type="int" indexed="true" stored="true" />
   <field name="SolrUpdatedTime" type="tdate" indexed="true" stored= "tr
ue" default="NOW+8HOUR" />

   <uniqueKey>PolicyID</uniqueKey>
```



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



属性说明如下。

- name：表示域名。
- type：表示域的类型，必须匹配类型，不然会报错。如果需要分词，那么就传分词器名，如 textPolicy_ik；另外，日期建议传 tdate，因为可以加快范围查找速度。
- indexed：是否要做索引。
- stored：是否要存储。
- required：是否必填。
- multiValued：是否有多个值。如果设置为多值，则里面的值就采用数组的方式来存储。

④ 对 data-config.xml 配置文件进行修改。

先注释掉默认的 dataConfig，然后在被注释内容的后面增加如下配置内容：

```
    <dataConfig>
        <dataSource driver="com.microsoft.sqlserver.jdbc.SQLServerDriver" url= "
jdbc:sqlserver://{SQLServer服务器IP地址}:{端口号，如果端口号是默认的1433，则可
不写};DatabaseName=SolrDB" user="sa" password="{登录SQL Server的密码}"/>
        <document name="Info">
          <entity name="Policy" dataSource="SolrDB" transformer= "ClobTra
nsformer" pk="PolicyID"
              query="SELECT [PolicyID], [PolicyGroupID], [PolicyOperatorI
D], [PolicyOperatorName], [PolicyCode], [PolicyName], [PolicyType], [Tic
ketType], [FlightType],  DATEADD(HOUR, 8, CAST([DepartureDate] AS DATETIM
E)) [DepartureDate], DATEADD(HOUR, 8, CAST([ArrivalDate] AS DATETIME)) [A
rrivalDate], DATEADD(HOUR, 8, CAST([ReturnDepartureDate] AS DATETIME))
[ReturnDepartureDate], DATEADD(HOUR, 8, CAST([ReturnArrivalDate] AS DATE
TIME)) [ReturnArrivalDate], [DepartureCityCodes], [TransitCityCodes], [A
rrivalCityCodes], [OutTicketType], [OutTicketStart], [OutTicketEnd], [Ou
tTicketPreDays], [Remark], [Status], DATEADD(HOUR, 8, CAST([SolrUpdatedT
ime] AS DATETIME)) [SolrUpdatedTime] FROM [Policy]"
              deltaImportQuery="SELECT [PolicyID], [PolicyGroupID], [PolicyO
peratorID], [PolicyOperatorName], [PolicyCode], [PolicyName], [PolicyType],
 [TicketType], [FlightType], DATEADD(HOUR, 8, CAST([DepartureDate] AS DAT
ETIME)) [DepartureDate], DATEADD(HOUR, 8, CAST([ArrivalDate] AS DATETIME))
 [ArrivalDate], DATEADD(HOUR, 8, CAST([ReturnDepartureDate] AS DATETIME))
 [ReturnDepartureDate], DATEADD(HOUR, 8, CAST([ReturnArrivalDate] AS DAT
ETIME)) [ReturnArrivalDate], [DepartureCityCodes], [TransitCityCodes],
[ArrivalCityCodes], [OutTicketType], [OutTicketStart], [OutTicketEnd],
```



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



```
[OutTicketPreDays], [Remark], [Status], DATEADD(HOUR, 8, CAST([SolrUpdat
edTime] AS DATETIME)) [SolrUpdatedTime] FROM [Policy] WHERE PolicyID = '
${dataimporter.delta.PolicyID}'"
          deltaQuery="SELECT [PolicyID] FROM [Policy] WHERE [SolrUpdatedT
ime] > '${dataimporter.last_index_time}'">
        <field column="PolicyID" name="PolicyID"/>
        <field column="PolicyGroupID" name="PolicyGroupID"/>
        <field column="PolicyOperatorID" name="PolicyOperatorID"/>
        <field column="PolicyOperatorName" name="PolicyOperatorName"/>
        <field column="PolicyCode" name="PolicyCode"/>
        <field column="PolicyName" name="PolicyName"/>
        <field column="PolicyType" name="PolicyType"/>
        <field column="TicketType" name="TicketType"/>
        <field column="FlightType" name="FlightType"/>
        <field column="DepartureDate" name="DepartureDate"/>
        <field column="ArrivalDate" name="ArrivalDate"/>
        <field column="ReturnDepartureDate" name="ReturnDepartureDate"/>
        <field column="ReturnArrivalDate" name="ReturnArrivalDate"/>
        <field column="DepartureCityCodes" name="DepartureCityCodes"/>
        <field column="TransitCityCodes" name="TransitCityCodes"/>
        <field column="ArrivalCityCodes" name="ArrivalCityCodes"/>
        <field column="OutTicketType" name="OutTicketType"/>
        <field column="OutTicketStart" name="OutTicketStart"/>
        <field column="OutTicketEnd" name="OutTicketEnd"/>
        <field column="OutTicketPreDays" name="OutTicketPreDays"/>
        <field column="Remark" name="Remark"/>
        <field column="Status" name="Status"/>
        <field column="SolrUpdatedTime" name="SolrUpdatedTime"/>
      </entity>
    </document>
  </dataConfig>
```

属性说明如下。

- query：查询数据库表中符合的记录数据。
- deltaImportQuery：表示次查询。次查询是获取以上步骤的 ID，然后获取其全部数据，根据获取的数据，对索引库进行更新操作，可能是删除、添加或修改。此查询只对增量导入起作用，可以返回多个字段的值，一般情况下，都是返回所有字段的列。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



- deltaQuery：查询需要增量索引的数据，所有经过修改的记录的 ID，可能是修改操作、添加操作或删除操作产生的。此查询只对增量导入起作用，而且只能返回 ID 值。

### 3. 增加 SolrUpdateTime 字段和触发器

为 SolrDB 数据库的 Policy 表增加字段和触发器，具体操作如下：

| 列名 | 数据类型 | 允许 Null 值 |
| --- | --- | --- |
| SolrUpdatedTime | datetime | ☑ |

```
USE [SolrDB]
GO

CREATE TRIGGER [dbo].[TR_Solr_UPDATE_Policy] ON [dbo].[Policy]
       FOR UPDATE, INSERT
AS
BEGIN
   IF UPDATE(PolicyID)
        OR UPDATE(PolicyGroupID)
        OR UPDATE(PolicyOperatorID)
        OR UPDATE(PolicyOperatorName)
        OR UPDATE(PolicyCode)
        OR UPDATE(PolicyName)
        OR UPDATE(PolicyType)
        OR UPDATE(TicketType)
        OR UPDATE(FlightType)
        OR UPDATE(DepartureDate) OR UPDATE(ArrivalDate)
        OR UPDATE(ReturnDepartureDate) OR UPDATE(ReturnArrivalDate)
        OR UPDATE(DepartureCityCodes)
        OR UPDATE(TransitCityCodes)
        OR UPDATE(ArrivalCityCodes)
        OR UPDATE(OutTicketType)
        OR UPDATE(OutTicketStart) OR UPDATE(OutTicketEnd)
        OR UPDATE(OutTicketPreDays)
        OR UPDATE(Remark)
        OR UPDATE(Status)
   BEGIN
              UPDATE dbo.Policy
              SET SolrUpdatedTime = GETDATE()
              FROM dbo.Policy p, inserted i
              WHERE p.PolicyID = i.PolicyID
```



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



```
    END
END
GO
```

#### 4. SolrNet

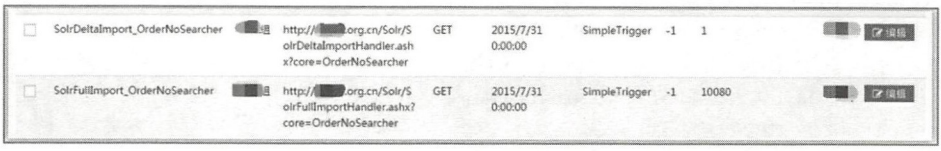
SolrNet 是 Solr 的开源.NET 客户端之一。

#### 5. 使用 Job 同步数据到 Solr

Solr 自身提供定时增量导入功能，但经测试，apache-solr-dataimportscheduler1.0 版本在 Solr 5.5 上已经不能使用，除非修改 apache-solr-dataimportscheduler 的源码。于是，我们采用了如下方式。

首先，开发 Job 任务调度 RESTful 服务，这种方式不仅可以实现定时增量数据导入，还可以实现定时全量数据导入。

然后，在自主研发的 Job 集中式管理平台中把相关内容都配置好，如下图所示。




这样，JobServer 就会定时以 HTTP GET、HTTP POST 或 HTTP HEAD 方式请求全量/增量导入链接，从而实现定时全量/增量数据导入功能。另外，如果想知道如何利用 SolrNet 实现全量导入、增量导入，则可以分别参考 Demo 代码中的 FullDataImport()和 DeltaDataImport()两个示例。

#### 6. 准实时数据导入、删除和查询

用 SolrNet 的 CURD API 实现，示例请见 Demo 的 Add()、Delete()和 Query()。准实时数据导入较定时增量数据导入更近于实时，在实际应用中，若通过消息队列对数据库和 Solr 同时进行更新，则效果更好。

## 12.3 Solr 的特性

- 具备高级全文搜索的能力；



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



- 高容量；
- 基于标准的开放接口（XML、JSON、HTTP）：Document 通过 HTTP 利用 XML 加到一个搜索集合中，查询该集合时也是通过 HTTP 收到一个 XML/JSON 响应来实现的；
- 提供功能全面的管理界面，使用户能够容易地控制 Solr 实例；
- 易监控；
- 高稳定性和容错性；
- 易配置，且不失灵活和适配性；
- 准实时索引，确保用户能够实时看到更新后的内容；
- 可扩展插件架构：新功能能够以插件的形式非常方便地添加到 Solr 服务器上。

## 12.4 Demo 下载

- SolrDemo 下载地址：https://github.com/das2017/SolrDemo



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 13 分布式协调器ZooKeeper

## 13.1 ZooKeeper是什么

Apache ZooKeeper是由Apache Hadoop的子项目发展而来的，于2010年11月正式成为Apache的顶级项目。

ZooKeeper是一个开放源代码的分布式协调服务。它具有高性能、高可用的特点，同时具有严格的顺序访问控制能力（主要是写操作的严格顺序性）。基于对ZAB协议（ZooKeeper Atomic Broadcast，ZooKeeper原子消息广播协议）的实现，它能够很好地保证分布式环境中数据的一致性。也正是基于这样的特性，使得ZooKeeper成为解决分布式数据一致性问题的利器。

## 13.2 ZooKeeper的工作原理简介

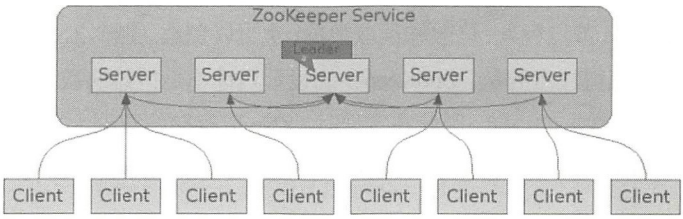
### 1. ZooKeeper的架构

ZooKeeper的整体架构如下图所示。

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






ZooKeeper 由两部分组成：ZooKeeper 服务端和客户端。

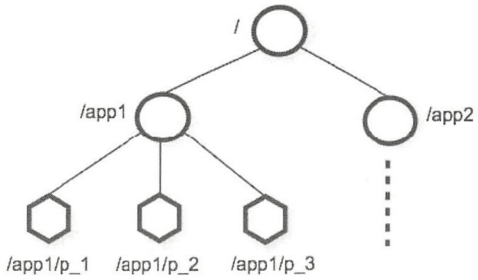
ZooKeeper 服务器采用集群的形式。值得一提的是，只要集群中存在超过一半的、处于正常工作状态的服务器，那么整个集群就能够正常对外提供服务。组成 ZooKeeper 集群的每台服务器都会在内存中维护当前的 ZooKeeper 服务状态，并且每台服务器之间都互相保持通信。

客户端在连接 ZooKeeper 服务集群时，会按照一定的随机算法选择集群中的某台服务器，然后和它共同创建一个 TCP 连接，使客户端连上那台服务器。而当那台服务器失效时，客户端会自动重新选择另一台服务器进行连接，从而保证服务的连续性。

当其中一个客户端修改数据时，ZooKeeper 会将修改同步到集群中所有的服务器上，从而使连接到集群中其他服务器上的客户端也能立即看到修改后的数据，很好地保证了分布式环境中数据的一致性。

### 2. ZooKeeper 的数据模型

ZooKeeper 的数据模型如下图所示。




ZooKeeper 的数据模型采用类似于文件系统的树结构。树上的每个节点称为 ZNode，而每个节点都可能有一个或多个子节点。ZNode 的节点路径标识方式是由一系列斜杠“/”进行分割的路径表示的。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



既可以向 ZNode 节点写入、修改和读取数据，也可以创建、删除 ZNode 节点或 ZNode 节点下的子节点。值得注意的是，ZooKeeper 的设计目标不是传统的数据库存储或大数据对象存储，而是协同数据的存储，因此在实现时，ZNode 存储的数据大小不应超过 1MB。另外，每一个节点都有一个 ACL（Access Control List，访问控制列表），据此控制该节点的访问权限。

ZNode 数据节点是有生命周期的，其生命周期的长短取决于数据节点的节点类型。节点类型共有 4 种：持久节点（PERSISTENT）、持久顺序节点（PERSISTENT_SEQUENTIAL）、临时节点（EPHEMERAL）和临时顺序节点（EPHEMERAL_SEQUENTIAL）。

### 3. Watcher：ZNode 数据变化通知

ZooKeeper 的 Watcher 机制概括为三个过程：客户端注册 Watcher 成为订阅者、服务端处理 Watcher 及客户端回调 Watcher。

客户端在自己需要关注的位于 ZooKeeper 服务器里的 ZNode 节点上注册一个 Watcher 监听后，一旦这个 ZNode 节点发生变化，则在该节点上注册过 Watcher 监听的所有客户端都会收到 ZNode 节点变化通知。在收到通知时，客户端通过回调 Watcher 做相应的处理，从而实现特定的功能。

## 13.3 ZooKeeper 的典型应用场景

通过对 ZooKeeper 中丰富的数据节点类型进行交叉使用，配合 Watcher 事件通知机制，可以非常方便地构建分布式应用中都会涉及的核心功能，如数据发布/订阅（即配置中心）、负载均衡、命名服务、分布式协调/通知、集群管理、Master 选举、分布式锁和分布式队列等。

### 1. 配置服务：ConfigServiceDemo

Demo 中的“ConfigServiceDemo（配置服务 Demo）”适用于 ZooKeeper 的配置中心应用场景：应用中用到的一些常用配置信息放到 ZooKeeper 的一系列 ZNode 节点上，供应用获取配置数据；同时，如果某应用在需要关注的配置项节点上注册了 Watcher，则以后每次被关注的配置项有更新时，都会实时通知该应用，从而达到获取最新配置信息的目的。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（1）为公司解决了什么问题。

① 减少了运维工作人员的工作量。

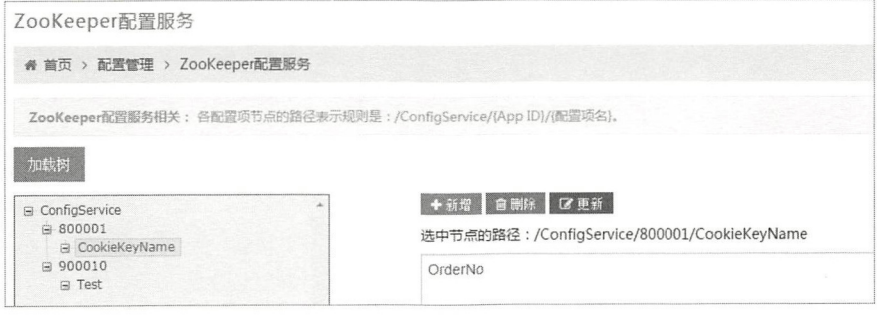
当公司的应用程序以集群环境模式被部署的时候，若是第 1 次部署应用程序或遇到需要配置要新增/修改/删除的情况，则运维工作人员不得不修改集群中的所有服务器。**而使用 ZooKeeper 后，他们只需要修改一次，就能为集群中的所有服务器完成配置的新增/修改/删除。**

② 使任意客户端能够看到即时生效的被改后的配置数据。

由于运维工作人员需要为集群中的所有服务器进行配置修改，从而导致了配置延时问题，使得集群中的每台服务器的配置数据不一致。也就是说，客户端（如应用程序）可能无法立即读取最新的配置值，需要过段时间后才能读取。当运维工作人员利用 ZooKeeper 修改配置数据后，新的配置数据会立即同步到集群中的所有服务器，从而保证集群中的所有服务器的配置数据对于任意客户端而言每时每刻都是准确无误的（可选加 Watcher）。

（2）ConfigService 管理。

下图显示的是 ZooKeeper 的配置服务页面。




## 2. Master 选举：MasterElectionDemo

集群中的服务器一般只有 1 台担任 Master 角色，一旦这台担任 Master 角色的服务器出现宕机情况，则会发生服务器单点故障问题。并且，我们可能不知道这台担任 Master 角色的服务器是从什么时候开始处于宕机状态的。利用 ZooKeeper 的“对于在 ZooKeeper 上创建的临时顺序节点（EPHEMERAL_SEQUENTIAL），一旦创建它的客户端与



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



ZooKeeper 服务集群之间的会话失效，那么该临时节点也就被自动清除”这一特性，再加上 Watcher 事件通知机制的使用，就能够解决服务器的单点故障问题——一旦当前作为 Master 角色的服务器宕机，那么它创建的临时顺序节点（EPHEMERAL_SEQUENTIAL）会马上消失；紧接着集群中注册过 Watcher 的所有服务器会马上收到当前 Master 服务器已宕机的通知，然后重新进行 Master 选举。

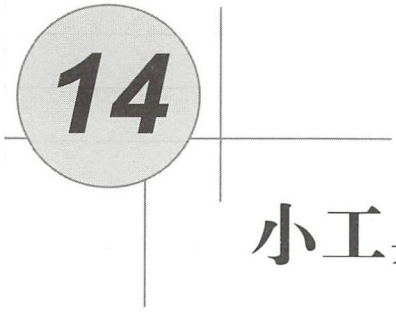
## 13.4 Demo 下载

- ZooKeeperDemo 下载地址：https://github.com/das2017/ZooKeeperDemo



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



# 14 小工具合集

## 14.1 ORM 工具

### 1. Dapper.NET 简介

Dapper.NET 是一个开源的轻型 ORM。它扩展了 IDbConnection 接口的功能，所以只要某个类实现 IDbConnection 接口，那么该类对象就能调用 Dapper.NET 中的方法。Demo 提供的 Dapper.dll 支持.NET Framework 4.0 版本及其以上版本。

### 2. 为什么选择使用 Dapper.NET

（1）语法简单，易学易用。

（2）无须依赖于具体的数据库工具，它能和所有.NET ado 提供商一起工作，如 MS SQL Server、Oracle、MySQL、PostgreSQL、SQLite、SQLCE 和 Firebird 等。

（3）运行速度快，接近于 IDataReader。因为它的映射工作原理是通过 Emit 反射 IDataReader 的序列队列来快速产生对象的。下面两张表格显示的数据（数据由官网提供）体现了它的性能优势。

Performance of SELECT mapping over 500 iterations - POCO serialization：

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



| 方　法 | 运行时间（单位：毫秒） |
| --- | --- |
| SqlDataReader | 47 |
| ExecuteMapperQuery | 49 |
| ServiceStack.OrmLite (QueryById) | 50 |
| PetaPoco | 52 |
| BLToolkit | 80 |
| SubSonic CodingHorror | 107 |
| NHibernate SQL | 104 |
| Linq 2 SQL | 181 |
| ExecuteStoreQuery | 631 |

Performance of SELECT mapping over 500 iterations - dynamic serialization：

| 方　法 | 运行时间（单位：毫秒） |
| --- | --- |
| ExecuteMapperQuery | 48 |
| Massive | 52 |
| Simple.Data | 95 |

### 3. 如何使用 Dapper.NET

提供的 Demo 都是关于 Dapper.NET 的基本用法。

首先，在需要用到 Dapper.NET 的项目中引用 Dapper.dll，如下图所示。然后在需要使用 Dapper.NET 的代码文件中加上“using Dapper;”。


提供的 Demo 包括如下主题：

（1）单条记录的增、改、删。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（2）批量记录的增、改、删。

（3）Query()泛型方法的使用。

（4）Query()非泛型方法的使用。

（5）QueryMultiple()方法的使用。

（6）ExecuteScalar()方法的使用。

（7）如何使用 Dapper.DynamicParameters 类。注意：当数据库表字段被设计为 char 类型时，必须给 DbType 传值，且必须赋的是 DbType.AnsiStringFixedLength，否则数据库的访问速度会突然变得很慢。

（8）如何调用存储过程。

## 14.2 对象映射工具

### 1. 为什么需要使用对象映射工具

比如，为了能够从数据库中获取数据，某一个基于 Windows Communication Service 的服务需要将数据库实体对象映射到数据协议对象上。对象—对象映射的一种传统做法是创建许多数据转换对象。这些对象负责在众多数据对象之间复制数据。对于拥有大量数据对象的程序而言，开发人员需要花费大量的时间精力编写大量的数据转换对象来支持数据对象映射。这一过程非常无聊沉闷，而且容易出现 Bug。如果使用对象—对象映射工具，则不需要编写那些数据转换对象。

### 2. EmitMapper 和 AutoMapper 简介

EmitMapper 和 AutoMapper 都是支持对象—对象映射的开源工具，主要负责将一个数据对象的数据映射到另外一个数据对象上。提供的 EmitMapper.dll 支持.NET Framework 3.5 及其以上版本；提供的 AutoMapper.dll 支持.NET Framework 4.5 及其以上版本。

### 3. EmitMapper 的使用方法

首先，在需要使用 EmitMapper 的项目中引用 EmitMapper.dll。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（1）基本使用方法。

采用默认的映射配置器 DefaultMapConfig 完成映射操作，不需要指定任何映射策略。写法主要如下（完整写法请见 BasicUsageDemo.cs）：

```
    ObjectsMapper<Source, Destination> mapper = ObjectMapperManager. Defa
ultInstance.GetMapper<Source, Destination>();
    Destination destination = mapper.Map(source);
```

或者：

```
    Destination destination = new Destination();
    ObjectMapperManager.DefaultInstance.GetMapper<Source, Destination>().
Map(source, destination);
```

默认的映射配置器能自动转换以下几种类型：

① 使用 ToString()方法将任何类型转换为 string 类型。

② 使用 System.Convert 类可使原生类型之间互相转换。

③ 可空类型转换为值类型或值类型转换为可空类型。

④ 枚举类型转换为它的基础类型或基础类型转换为对应的枚举类型。

⑤ 枚举类型转换为 string 类型或 string 类型转换为枚举类型。

⑥ 不同的集合类型之间互相转换（如 Array、ArrayList、List<>、IEnumerable）。

⑦ 类转换为结构或结构转换为类。

⑧ 具有内嵌类型成员的复杂类型采用递归方式转换。

（2）使用 DefaultMapConfig 的自定义配置方法。

如果默认的转换满足不了需求，则可考虑调用 DefaultMapConfig 提供的配置方法。下表说明了各配置方法的作用。

| 配置方法名 | 描　述 |
| --- | --- |
| ConstructBy<T> | 让目标对象使用指定的构造函数替代默认构造函数 |
| ConvertUsing<From, To> | 为指定的非泛型成员提供自定义的转换逻辑 |



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



续表

| 配置方法名 | 描　述 |
| --- | --- |
| ConvertGeneric | 为指定的泛型成员提供自定义的转换逻辑 |
| ShallowMap、 ShallowMap<T> | 指定的复杂类型成员采用复制引用的映射方式，即浅映射。默认为浅映射 |
| DeepMap、 DeepMap<T> | 映射时，目标对象另被构造，即深映射。若有内嵌类型的成员，则也是采用这种方式被映射的 |
| IgnoreMembers<TFrom, TTo> | 忽略指定成员的映射 |
| MatchMembers | 如果源对象的成员名和目标对象的成员不匹配，则映射时，自定义成员名匹配逻辑传给此配置方法，使源对象的成员名能够映射到目标对象匹配的成员名 |
| NullSubstitution<TFrom, TTo> | 源对象的成员值为 NULL。用此配置方法映射后，目标对象对应的成员值是其他值，替代 NULL |
| PostProcess<T> | 映射完成后执行指定的方法 |

### 4. AutoMapper 的使用方法

在需要使用 AutoMapper 的项目中引用 AutoMapper.dll。

提供的 Demo 主要包括如下主题：

- 最基本的用法（写了 3 种）；
- 扁平化映射；
- 前后映射；
- 空值替换；
- 忽略映射；
- 条件映射；
- 指定映射字段；
- 强类型对象映射动态对象；
- 动态对象映射动态对象；
- 自定义类型转换器；



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



- 自定义解析器。

### 5. EmitMapper 和 AutoMapper 的优缺点

EmitMapper 和 AutoMapper 各有千秋。

EmitMapper 官网上虽然有多年的时间没有更新，但它的性能却十分高（接近硬编码）。下图显示的是通过笔者 PC 运行出来的结果，发现 EmitMapper 的映射速度比 AutoMapper 的映射速度快得多（被比较的 AutoMapper 版本号是 5.1.1）。

```
Handwritten Mapper: 26 milliseconds
Emit Mapper: 53 milliseconds
Auto Mapper: 440 milliseconds
```

AutoMapper 虽然性能比不过 EmitMapper，但官网上一直保持着更新状态。

## 14.3 IoC 工具

### 1. Autofac 简介

Autofac 是一款轻量级的开源 IoC 容器，它主要负责管理类之间的依赖关系和管理对象的生命周期等，降低应用程序组件间的耦合性，提高类、组件的扩展性和可重用性。

### 2. 背景

在软件系统中，通常都是通过很多对象（系统、模块、对象）的共同协作来最终实现业务系统的。很多对象的协作肯定会产生或多或少的耦合（依赖），降低对象之间的耦合是软件工程永远追求的目标之一。

### 3. 依赖倒置原则

- 上层模块不应该依赖于下层模块，它们应该共同依赖于一个抽象。
- 抽象不应该依赖于具体，具体可以依赖于抽象。

### 4. IoC

Inversion of Control：控制反转，反转的是依赖对象的控制权。

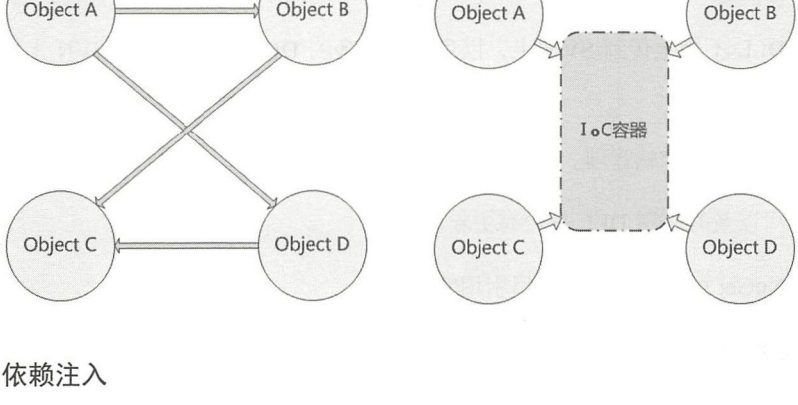


仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



如果 A 依赖 B，则按照之前的做法是在类 A 中需要 B 的地方主动实例化一个对象 B。现在的做法是类 A 中需要一个对象 B，IoC 容器初始化一个对象 B 传给类 A。创建依赖对象的职责从类 A 转移到了 IoC 容器中，如下图所示。




### 5. 依赖注入

可以用不同的方式实现 IoC，其中一种实现策略是依赖注入。那么依赖注入是什么？把耦合从代码中转移到配置文件中，通过一个 IoC 容器，在需要的时候再形成这个依赖关系，即在程序中把需要的接口实现注入需要它的类中，这就是依赖注入。

### 6. 优点

- 可维护性好：在通过 IoC 容器创建组件之间的依赖关系之前，这些组件之间是毫不相关的，都是独立的单元，便于各自调试和单元测试。
- 分工明确、提高开发效率：各个组件都是独立的单元，可以由不同的开发团队来开发和维护，大大提高了开发效率。
- 可重用性高：常用的模块都是一个单独的个体，实现了标准的接口，可以插接到任何支持此标准的模块中。

## 14.4 DLL 包管理工具

### 1. NuGet 简介

NuGet 是 Visual Studio 的一个扩展。在使用 Visual Studio 开发基于.NET Framework



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



的应用时，NuGet 能把在项目中添加、移除和更新引用的工作变得更加快捷方便。

### 2. 为什么要用 NuGet

（1）由于公司内部的公共组件越来越多，为了统一管理这些公共组件，需要搭建公司内部的 NuGet 服务器。

（2）DLL 不用上传到 SVN 上，以免造成过多的 DLL 文件被传到 SVN 上增加 SVN 压力。

（3）方便包的依赖管理。

（4）可以及时知道 DLL 是否有更新。

（5）NuGet 可以自动还原项目引用的包。

### 3. 使用方法

（1）设置 NuGet 服务器。

单击鼠标右键选择需要添加引用的项目文件→管理 NuGet 程序包，便打开了如下图所示的弹出框，然后单击“设置”按钮。

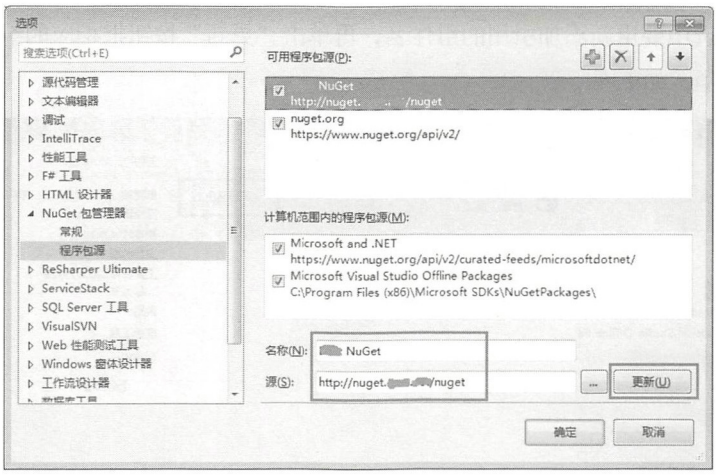



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



添加程序包源，即添加公司内部的 NuGet 服务器名及其地址（http://nuget.***.***/nuget），如下图所示。




（2）添加组件引用。

在左侧的联机列表中，选中在上一步设置的 NuGet 服务器名（*** NuGet），然后在中间的列表中选中要添加的引用的组件名，再单击“安装”按钮把相应的组件引用添加到项目中，如下图所示。



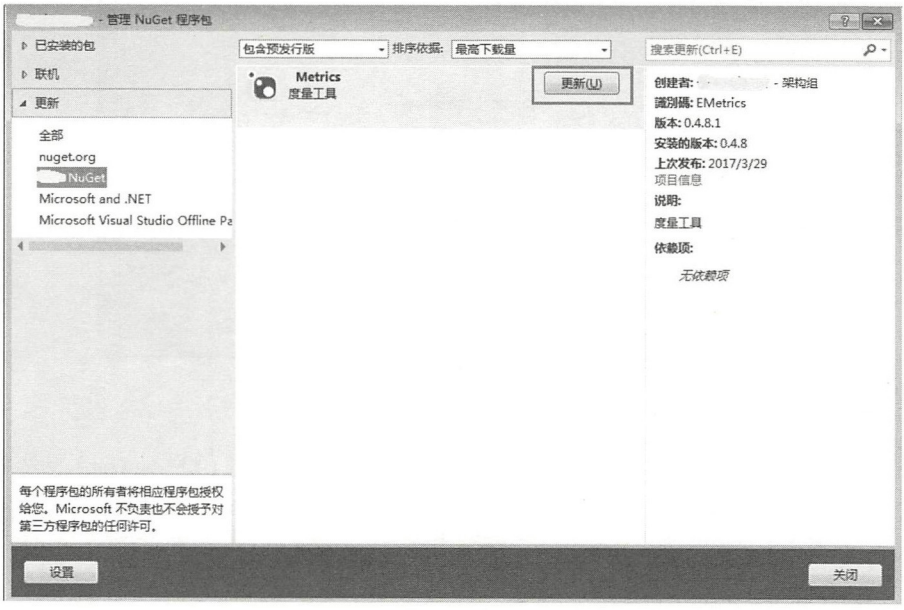

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（3）更新组件引用。

在更新列表中，选中在前面步骤中设置的 NuGet 服务器名（*** NuGet），然后在中间的列表中选中要重新添加引用的组件名，再单击“更新”按钮把相应的组件引用重新添加到项目中，如下图所示。

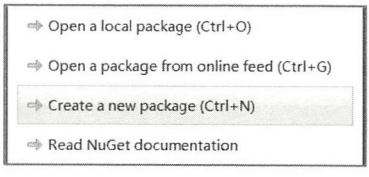



（4）包管理。

管理包时需要用到 NuGetPackageExplorer，下载地址见 14.5 节。

① 新建包。

步骤 1：打开 NuGet Package Explorer，单击“Create a new package(Ctrl+N)”（创建一个新的组件包），如下图所示。



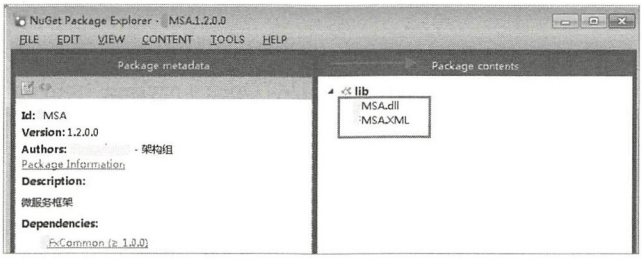

步骤 2：将需要打包的组件引用拖放到“Package contents”区域，如下图所示。



仅供非商业用途或交流学习使用

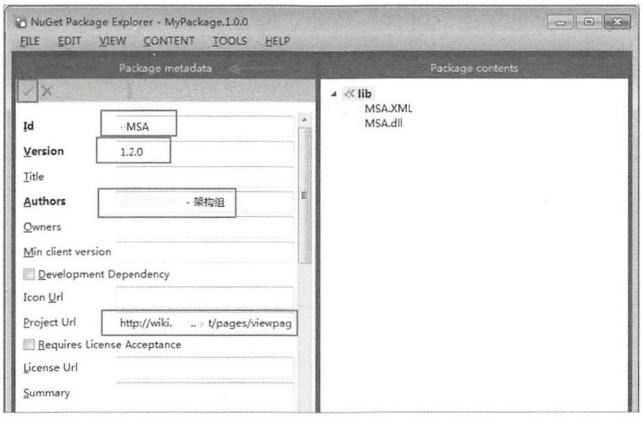
非卖品，仅供非商业用途或交流学习使用





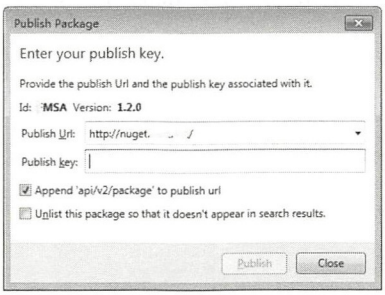

单击位于上图左上角的【Edit Metadata】按钮后，进入如下图所示的编辑界面。

在【Package metadata】区域中编辑好组件的相应信息，然后勾选绿色的“对号”，单击 FILE→Save 保存；其中，包名（即包 Id 号）的命名规范建议是{产品线英文名全称}.{AppID}.{***}。




步骤 3：组件发布。

单击“FILE”→“Publish”后，弹出如下图所示的对话框，在 Publish Package（发布地址）中输入：http://nuget.***.***/，在 Publish Key 文本框处输入密码。





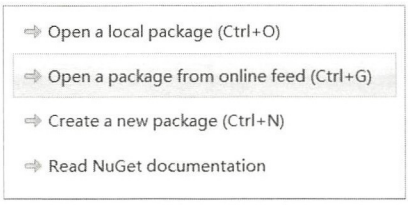

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

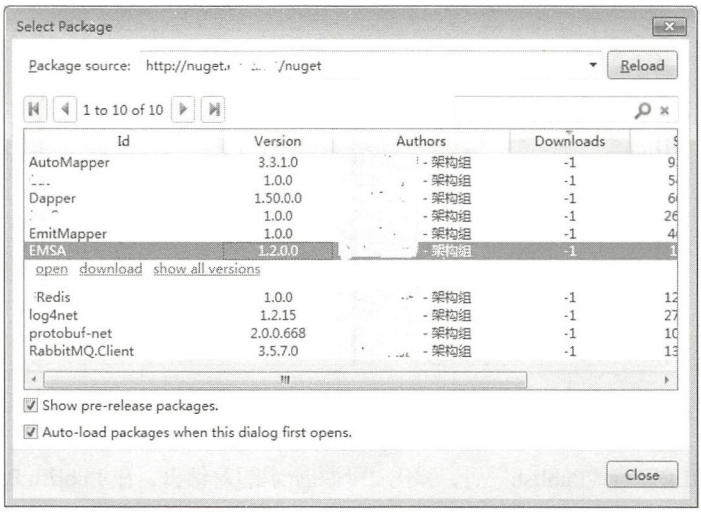


② 更新包。

步骤 1：打开 NuGet Package Explorer，单击从“Open a package from online feed(Ctrl+G)”（从在线源中打开一个包），如下图所示。




步骤 2：在 Package source 文本框处默认显示了“http://nuget.***.***/nuget”，即公司内部的 NuGet 服务器地址，再单击“Reload”按钮。然后，在出现的包列表中选中要编辑的包。最后，双击它或单击“open”按钮，如下图所示。




步骤 3：单击位于界面左上角的“Edit Metadata”按钮后，进入如下图所示的编辑界面。

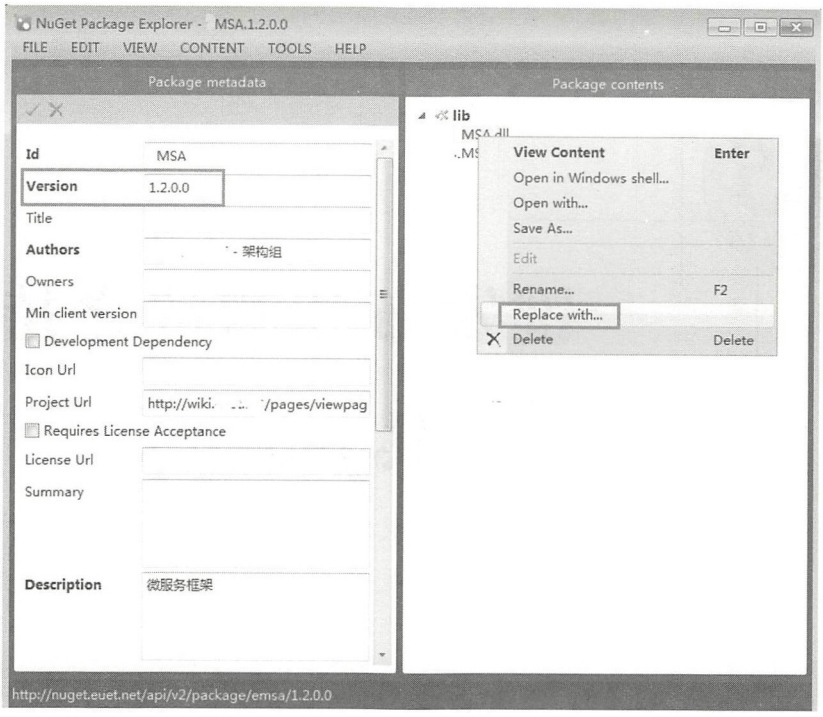
在编辑界面的“Package metadata”区域中，在“Version”文本框中增大版本号。然后，在编辑界面的“Package contents”区域中，右键单击需要更新的引用，在弹出的快捷菜单中单击“Replace with...”来完成重新上传最新的包的操作。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






步骤 4：编辑完成之后，单击位于上图左上角的绿色“对号”，然后单击“FILE”→“Publish”进行发布工作。

## 14.5 Demo 下载

- Dapper.NETDemo 下载地址：https://github.com/das2017/DapperDemo
- EmitMapperDemo 和 AutoMapperDemo 下载地址：https://github.com/das2017/ObjectMapperDemo
- AutofacDemo 下载地址：https://github.com/das2017/AutofacDemo



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 15 一键发布和测试之持续集成工具 Jenkins

## 15.1 Jenkins 简介

当每月发布次数变得越来越多时，如超过 500 次，则发布工作人员的工作量会翻倍增长，此时由人工发布操作失误引起的风险会变得越来越大。为了提高项目的发布效率，也为了降低由人工操作失误带来的风险，需要引进持续集成工具。

Jenkins 是一个用 Java 语言编写的开源的持续集成工具，最开始被称为 Hudson。Jenkins 在持续集成领域市场份额中居于主导地位，被各种规模的团队用于用各种语言实现的各类项目中。例如，C#、Java、Ruby、Groovy、Grails、PHP 等。选择 Jenkins 的理由如下。

（1）**易于使用**：Jenkins 的用户界面简单、直观、友好，发布工作人员只需通过简单的 UI 操作就可以替代原来烦琐的发布工作。

（2）**拥有良好的扩展性**：有数以百计的开源插件可供使用，而且几乎每周都会有新的开源插件贡献进来，这些插件的安装都十分快捷和简单。

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

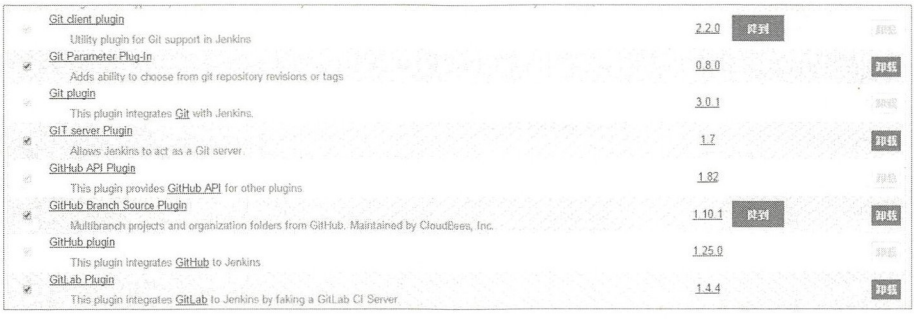


（3）**受大众喜欢**：Jenkins 开源社区的规模变得越来越大，活跃度也变得越来越高，发展速度非常快。

## 15.2 Jenkins 插件与相关工具

（1）Jenkins：持续集成工具。

（2）Git：源代码管理工具，是目前流行的分布式版本控制系统。需要安装的 Jenkins 插件如下图所示。




（3）TFS：可选，源代码管理工具。

（4）MSBuild：Visual Studio 中自带的一个程序编译组件。需要安装的 Jenkins 插件是 MSBuild Plugin。

（5）FTP：可选，通过 FTP 把编译好的发布文件部署到应用服务器中。需要安装的 Jenkins 插件是 Publish Over FTP。

（6）Jenkins 角色及权限管理：需要安装的 Jenkins 插件是 Role-based Authorization Strategy。

（7）Python 脚本：自己编写的 Python 脚本放在 Jenkins 服务器中。可以实现 Jenkins 把编译好的发布文件部署到远程应用站点服务器，以及实现回滚操作（Rollback）。

（8）PxExec.exe 工具：装在 Jenkins 服务器中，利用这个工具，可以在远程服务器中执行命令，如 xcopy。

（9）SoapUI 自动化测试：用于接口测试自动化，同时需要安装的 Jenkins 插件是 HTML Publisher plugin。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（10）回滚操作（Rollback）：需要安装的 Jenkins 插件是 Build with Parameters，用于指定哪个项目回滚到哪个备份版本。

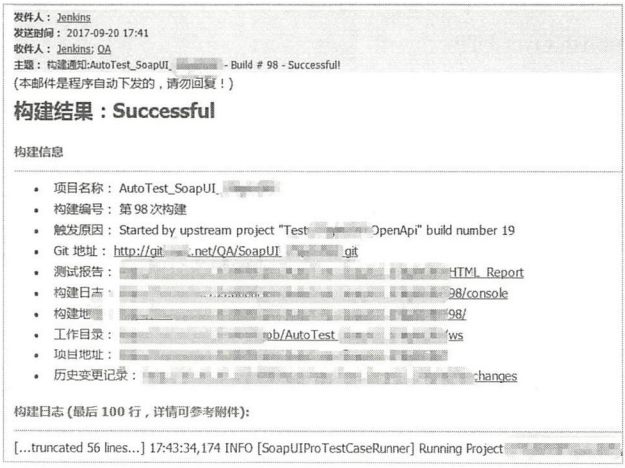
## 15.3 Jenkins 关键配置

### 1. 邮件配置

E-mail 是 Jenkins 的基本通知技术。什么情况下需要 Jenkins 发送电子邮件通知？例如，在一次构建失败（编译错误）后。

邮件配置如下图所示。


邮件通知结果如下图所示。





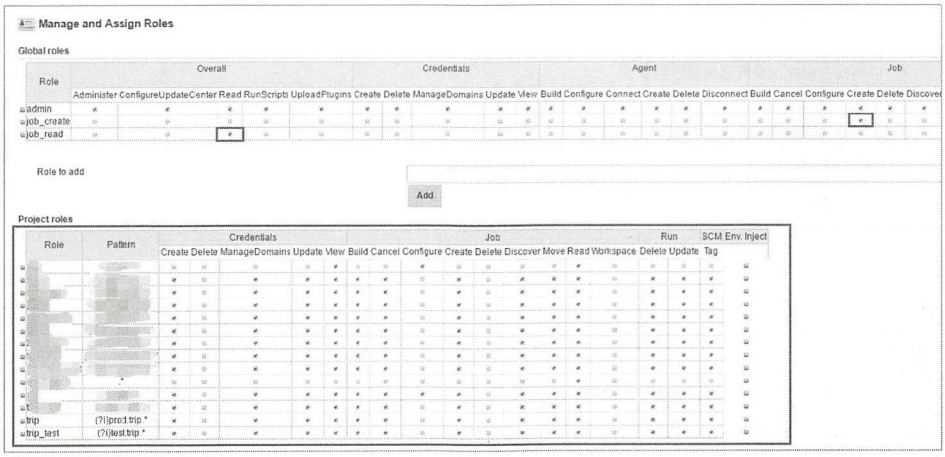

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

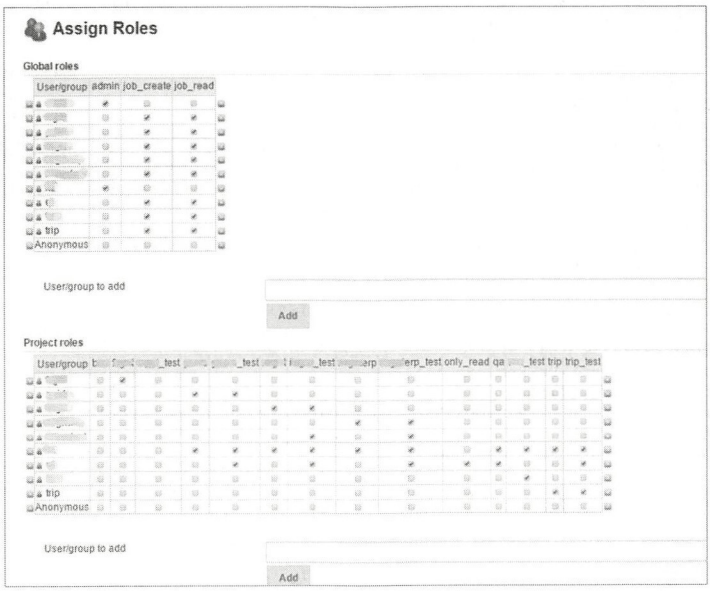


## 2. 角色及权限管理

首先设置全局角色和项目角色，其中 Pattern 是用来设置构建作业名的命名规范。例如，规定了构建作业名的命名规范是{发布环境}.{产品线英文名全称}.{项目名}，那么要发到生产环境、属于 Trip 产品线的所有构建作业，其 Pattern 设置的值为“(?i)prod.trip.*”，表示构建作业名必须以 prod.trip 开头，而且不区分大小写，用于发布到生产环境，如下图所示。




然后，分别为 Jenkins 账号分派全局角色和项目角色，如下图所示。





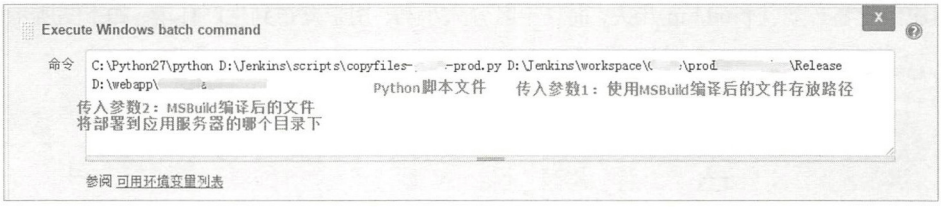

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 3. 部署到集群

Jenkins 通过运行自己编写的 Python 脚本把编译好的发布文件部署到远程应用站点服务器，同时同步到集群内其他应用站点服务器，所以需要新增构建步骤配置，如下图所示。




其中，.py 脚本（即 Python 脚本）的内容如下图所示。

```
# -*- coding: utf-8 -*-
import os
import sys
from datetime import datetime
import zipfile
import shutil
#程序入口
if __name__ =="__main__":
    os.system('''net use * /delete /y''')
    a=datetime.now()
    #按照日期取名备份
    output_filename="D:\\webbackup\\"+
    os.path.basename(sys.argv[2])+"-"+a.strftime('%Y%m%d-%H%M')+"\\"
    print "backuping..."
    bak="C:\\test\\PsExec.exe \\\\ . . .  -u  \\administrator -p
    xcopy "+sys.argv[2]+"  "+output_filename+" /Y /E /S "
    #远程应用服务器进行备份操作
    os.system(bak)
    print "backup finish"
    #挂载远程应用服务器共享
    os.system('''net use W: \\\\ . . . \\webapp passwd /user:''')
    os.system('''net use X: \\\\ . . . \\webbackup passwd /user:''')
    #拷贝位于Jenkins服务器的已编译好的文件到位于远程应用服务器的临时文件夹之下
    os.system("xcopy "+sys.argv[1]+" X:\\tmp\\"+os.path.basename(sys.argv[2])+"\\ /Y /E
    /S /d  /exclude:D:\Jenkins\scripts\\nocopy.txt")
    os.system('''net use * /delete /y''')
    #在该远程应用服务器中，把临时文件夹下的文件，复制到Web站点目录下
    sync="C:\\test\\PsExec.exe \\\\ . . .  -u  \\administrator -p
    xcopy D:\\webbackup\\tmp\\"+os.path.basename(sys.argv[2])+"
    D:\\webapp\\"+os.path.basename(sys.argv[2])+"\\ /Y /E /S /d"
    os.system(sync)
    #同步Web站点目录下的文件到位于另一台远程应用服务器的Web站点目录下
    to_ ="C:\\test\\PsExec.exe \\\\ . . .  -u  \\administrator -p
    xcopy "+sys.argv[2]+"
    \\\\ . . . \\webapp\\"+os.path.basename(sys.argv[2])+"\\ /Y /E /S /d"
    os.system(to_ )
    os.system('''net use * /delete /y''')
```

填的是具备读写权限的本地账户

.py 脚本实现了如下逻辑。

- **第 1 步备份**：在远程应用站点服务器中，备份要部署新版本的应用系统的所有



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

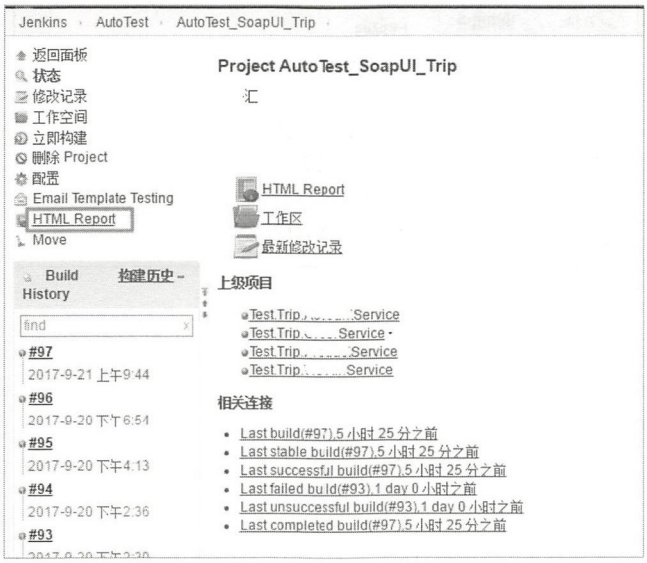


文件。利用 PxExec.exe 工具，让 Jenkins 服务器远程连到应用站点服务器。然后在这台应用站点服务器中，利用 xcopy 命令，将要部署新版本的应用系统站点目录下的所有文件复制到这台应用站点服务器中的备份目录下。

- **第 2 步部署**：部署到这台远程应用站点服务器。先利用 xcopy 命令，把由 Jenkins 编译好的、位于 Jenkins 服务器的文件复制到这台应用站点服务器共享目录下，以.config 结尾的配置文件不会被复制。再利用 PxExec.exe 工具，让 Jenkins 服务器远程连到这台应用站点服务器。然后在这台应用站点服务器中，利用 xcopy 命令，把临时存放目录下的文件复制到这个应用系统站点目录下。
- **第 3 步同步**：同步发布文件到该应用系统集群内的其他应用站点服务器。利用 PxExec.exe 工具，让 Jenkins 服务器远程连到这台应用站点服务器，然后在这台应用站点服务器中，利用 xcopy 命令，把该应用系统站点目录下的所有文件复制到集群内的其他应用站点服务器的该应用系统站点目录下。

### 4. 接口自动化测试 SoapUI

测试用例提交到版本库（如 Git）后，可通过 Jenkins 对其进行编译，编译后，通过 SoapUI 一键调用，开始自动化测试。一旦自动化测试完成，就会生成一份报表，通过 HTML Report 呈现出来，如下面两张图所示。

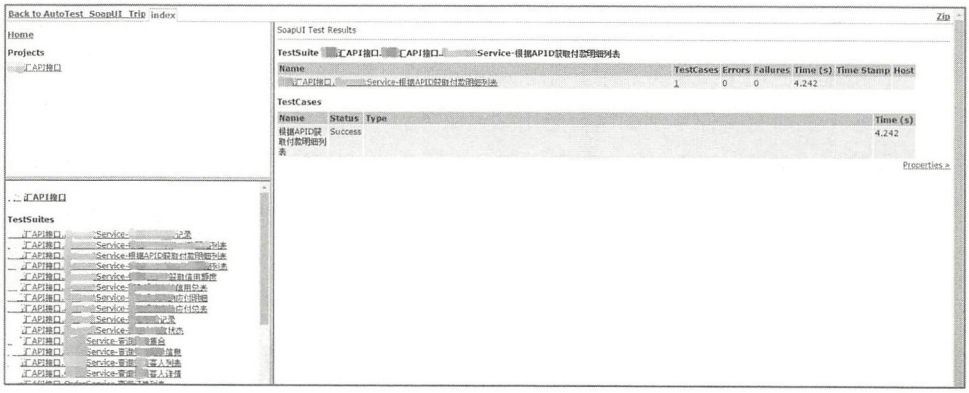





仅供非商业用途或交流学习使用

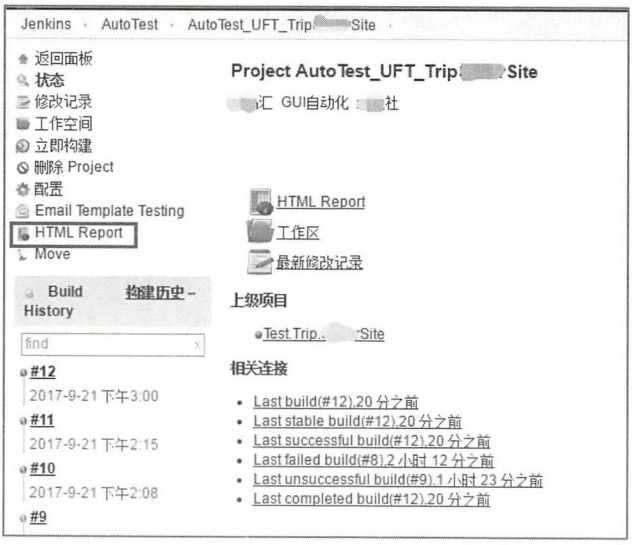
非卖品，仅供非商业用途或交流学习使用






## 5. 界面自动化测试 UFT

自动化测试结果如下面两张图所示。

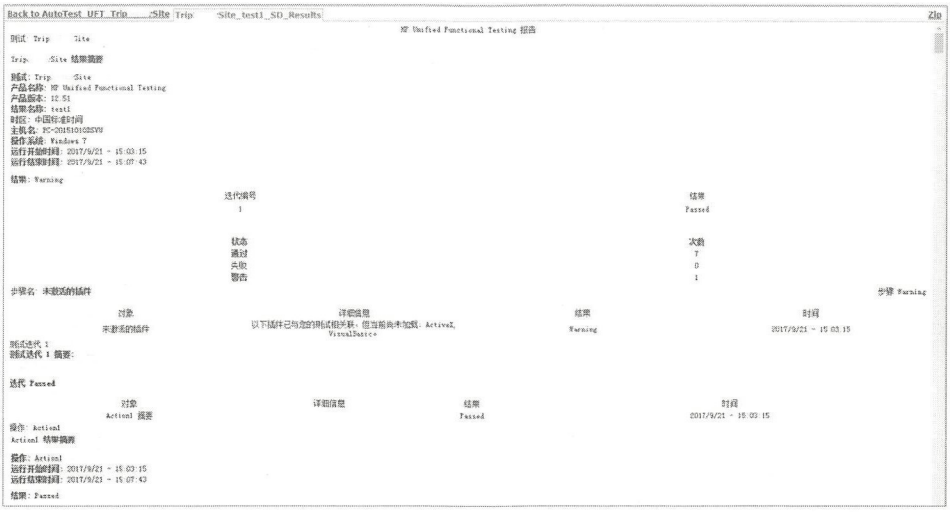





仅供非商业用途或交流学习使用

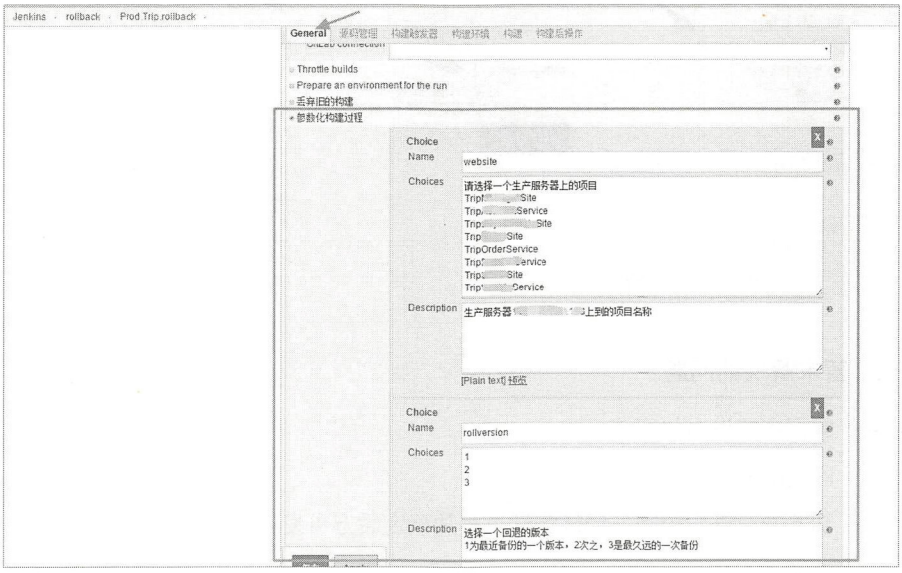
非卖品，仅供非商业用途或交流学习使用






## 6. 回滚操作（Rollback）

General 的配置如下图所示。




构建的配置如下图所示。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用




选择将要回滚哪个项目及回滚到哪个备份版本号，如下图所示。


## 7. 暂未解决的问题

- 数据库的发布与回滚；
- 应用配置文件的发布与回滚；
- 加入 QA 流程控制，经过测试工程师确定后方可发布。

以上三个问题也可以借助其他工具来实现，分别是数据库发布工具、集中式配置服务、流程管理工具，甚至邮件确认。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



## 15.4 Jenkins 的使用价值

（1）减少发布工作人员的大量日常工作量，大大提高项目的发布效率。

（2）不容易出错，降低人工发布带来的风险。

（3）可 24 小时随时发布。

（4）方便紧急修复或回滚操作。

（5）方便对发布流程进行控制，实现标准化。

（6）方便发布统计，历史版本可追溯。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 第4篇

# 公共应用篇

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 16 单点登录

## 16.1 单点登录简介

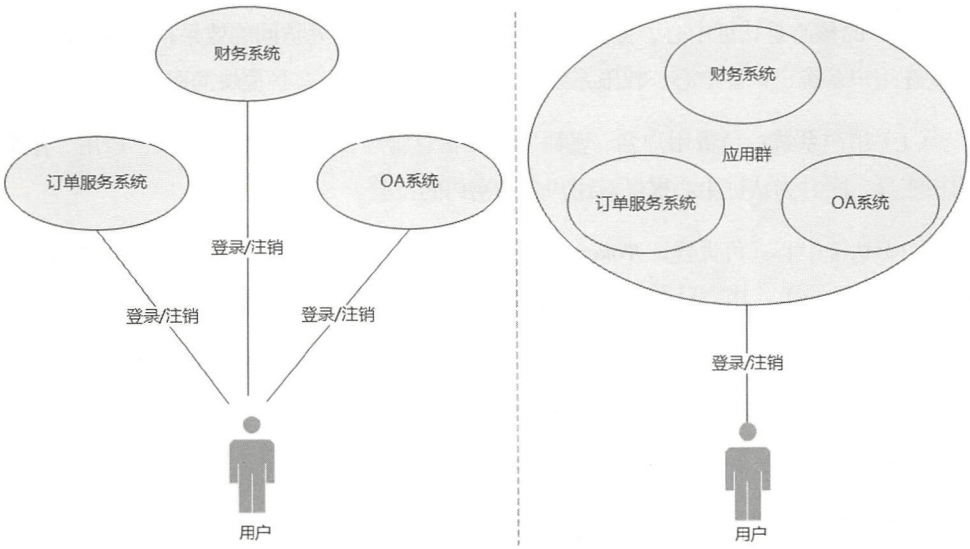
比如有这样的一个场景：公司内部有财务系统、OA 系统和订单服务系统等各类相互独立的应用系统，员工张三有财务系统、OA 系统和订单服务系统的操作权限。当公司内部有 100 个应用系统时，张三是不是要输入 100 次用户名和密码进行登录，然后才能进行业务操作呢？显然这是很不好的体验，因此我们需要引入一个这样的机制：张三只需输入一次用户名和密码登录成功后，就可以访问财务系统、OA 系统和订单服务系统，这就是单点登录。

单点登录的英文全称是 Single Sign On，简称 SSO。即用户只需要登录一次，就可以在个人权限范围内，访问所有相互信任的应用功能模块，不管整个应用群的内部有多么复杂，对用户而言，都是一个统一的整体。用户访问 Web 系统的整个应用群与访问单个系统一样，登录和注销分别只要一次就够了。单点登录的示意图如下图所示。

仅供非商业用途或交流学习使用

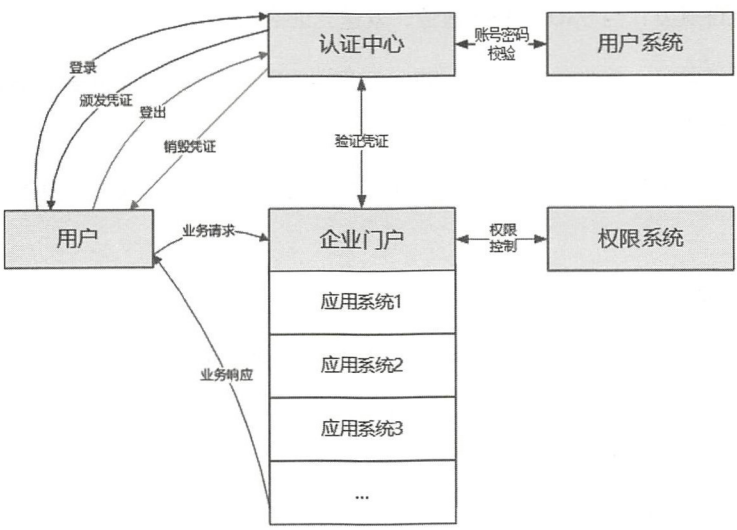
非卖品，仅供非商业用途或交流学习使用






## 16.2 SSO 技术实现

单点登录要想实现好并不容易，下图是我们的具体实现。SSO 需求优先级首先是单点登录和单点注销功能，然后是应用接入的门槛，最后是数据安全性，安全性对于 SSO 也非常重要。






仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



SSO 的核心是认证中心，但要实现用户一次登录、到处访问的效果，技术实现需要建立在用户系统、认证中心、权限系统、企业门户的基础上，各模块的职责如下。

（1）用户系统：负责用户名、密码等账户信息管理，包括增加、修改、启用、停用用户账号，同时为认证中心提供对用户名和密码的校验。

（2）认证中心：负责凭证 Token 的生成、加密、颁发、验证、销毁、登入（Login）和登出（Logout）。用户只有在拥有凭证并验证通过的情况下才能访问企业门户。

（3）权限系统：负责角色管理、资源设置、授权设置、鉴定权限，具体实现可参考 RBAC。权限系统可为企业门户提供用户权限范围内的导航。

（4）企业门户：作为应用系统的集成门户（Portal），集成了多个应用系统的功能，为用户提供链接导航、用户信息和登出功能等。

### 1. SSO 服务端功能

（1）登录认证：接收登录账号信息，让用户系统验证用户的登录信息。

（2）凭证生成：创建授权凭证 Token。生成的凭证一般包含用户账号、过期时间等信息，凭证是一串加密的字符串，如 AES 加密（凭证明文+MD5 加密信息），可采用 JWT 标准。

（3）凭证颁发：与 SSO 客户端通信，发送凭证给 SSO 客户端。

（4）凭证验证：接收并校验来自 SSO 客户端的凭证有效性，凭证验证包括算法验证和数据验证两种。

（5）凭证销毁与登出：接收来自 SSO 客户端的登出请求，记录并销毁凭证，跳转至登录页面。

### 2. SSO 客户端功能

（1）请求拦截：拦截应用的未登录请求，跳转至登录页面。

（2）获取凭证：接收并存储由 SSO 服务端发来的凭证，凭证传输的方式有 HTTP Header、Cookie 和网址参数等。

（3）提交凭证验证：与 SSO 服务端通信，发出校验凭证有效性的请求。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

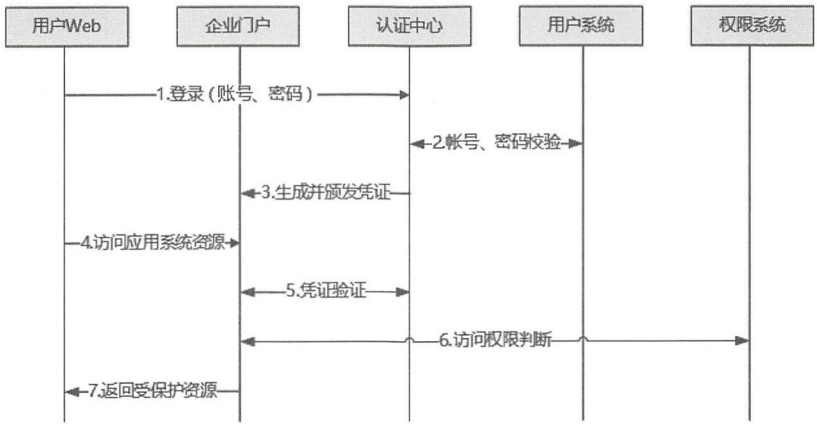


（4）获取用户权限：获取该凭证的用户权限，并返回受保护资源给用户。

（5）凭证销毁与登出：销毁本地会话，然后跳转至登出页面。

### 3. 用户单点登录流程

登录流程如下图所示。




（1）登录：将用户输入的用户名和密码发送至认证中心，然后认证中心调用用户系统来验证登录信息。

（2）生成并颁发凭证：通过登录信息的验证后，认证中心创建授权凭证 Token，然后把这个授权凭证 Token 返回给 SSO 客户端。SSO 客户端获取这个 Token，进行存储。在后续请求中，在 HTTP 请求数据中都得加上这个 Token。

（3）凭证验证：SSO 客户端发送凭证 Token 给认证中心，认证中心校验这个 Token 的有效性。凭证验证有算法验证和数据验证，算法验证可在 SSO 客户端完成。

### 4. 用户访问和单点注销

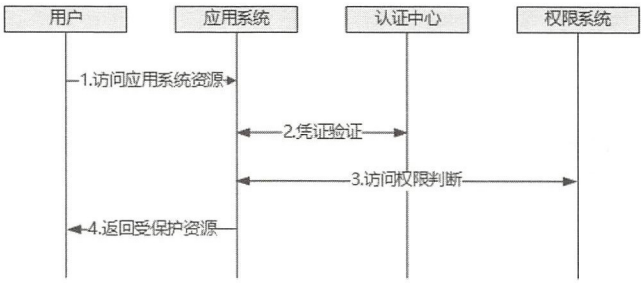
（1）用户访问：用户如果没有有效的凭证，则认证中心将强制用户进入登录流程，下图是用户访问的流程。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

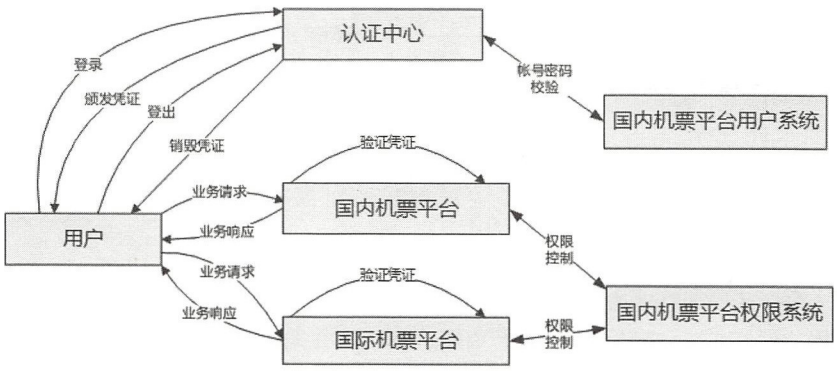





（2）单点注销：用户如果注销了应用群内的其中一个应用，则全局 Token 也会被销毁，应用群内的所有应用将不能再被访问。

### 5. 应用接入与集成

流程如下图所示。




我们的应用接入与集成具体如下。

（1）用户系统：接入国内机票平台的用户系统，负责登录认证。

（2）权限系统：接入国内机票平台的权限系统。

（3）认证中心：负责生成并颁发凭证、销毁凭证，改造国内机票平台的登录、登出。

（4）凭证验证：在国内机票、国际机票应用系统中调用 SSO 客户端组件实现凭证的验证。

（5）企业门户：由国内机票平台承担。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



## 16.3 JWT 规范

JSON Web Token（JWT）是目前应用最为广泛的 Token 格式，是为了在网络应用环境间传递声明而执行的一种基于 JSON 的开放标准（RFC 7519）。该 Token 被设计为紧凑且安全的，特别适用于分布式站点的单点登录、API 网关等场景。JWT 的声明一般被用来在身份提供者和服务提供者之间传递被认证的用户身份信息，以便从资源服务器获取资源，也可以增加一些额外的其他业务逻辑所必需的声明信息。该 Token 可直接被用于认证，也可被加密。JWT 信息体由 3 部分构成：头（Header）+载荷（Payload）+签名（Signature），具体优点如下：

- JWT 支持多种语言，C#、Java、JavaScript、PHP 等很多语言都可以使用。
- JWT 可以自身存储一些和业务逻辑有关的必要的非敏感信息，因为有了 Payload 部分。
- 利于传输，因为 JWT 的构成非常简单，字节占用很小。
- 不需要在服务端保存会话信息，不仅省去服务端资源开销，而且易于应用的扩展。



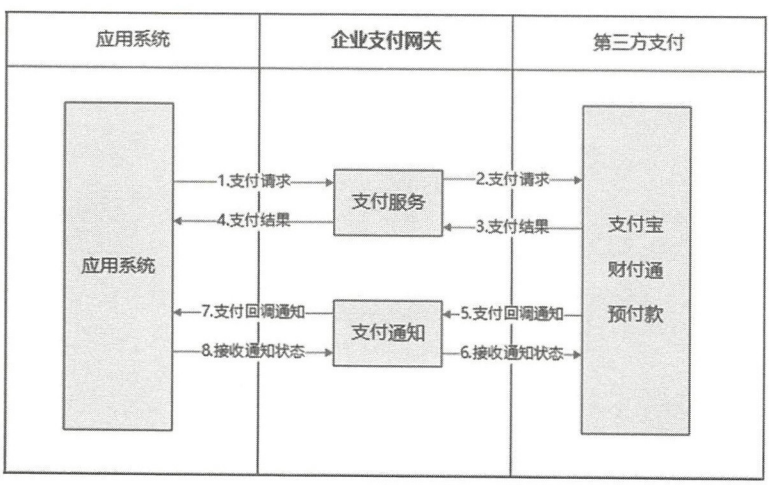
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 17 企业支付网关

## 17.1 企业支付网关介绍

企业支付网关交互如下图所示。




企业支付网关又叫聚合支付，由统一支付服务、统一支付通知和统一支付后台三部分组成，本章我们主要介绍前两部分。将企业支付网关独立出来非常有必要，它是企业

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



未来金融事业部的基础，当前价值也很大，具体如下。

（1）集中研发工作：集中研发封装公司使用的各种支付方式，如支付宝、财付通和预付款等，同时统一各应用系统的支付调用方式。

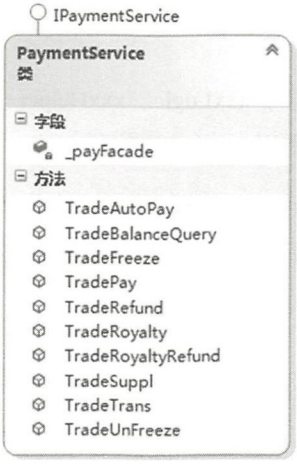
（2）集中运维工作：集中发布、监控、维护和安全等工作。

（3）集中财务工作：集中各支付方式的对账、统计、日志审计和异常处理等后台运营工作。

## 17.2 统一支付服务

### 1. 统一支付接口

统一支付接口如下图所示。




以上接口有支付、代扣、分润、退款、退分润、补差、转账、冻结、解冻和预付款。支付接口服务仅负责生成和返回支付链接，由调用的业务应用来负责 URL 跳转。

### 2. 统一支付架构

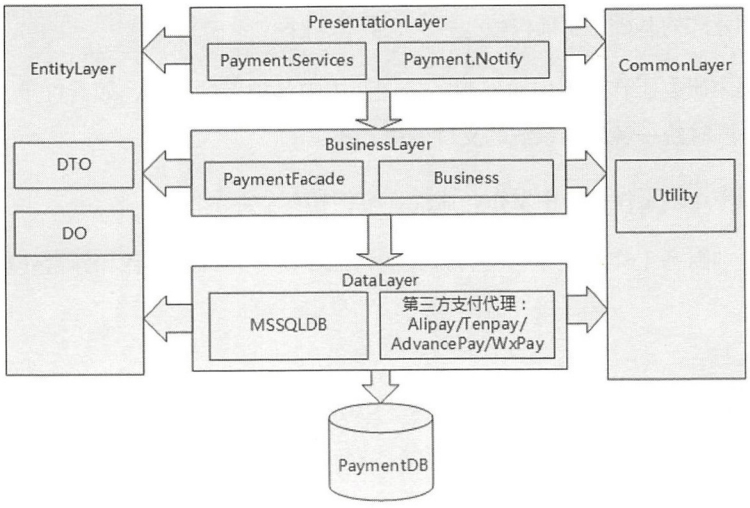
统一支付架构如下图所示。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用





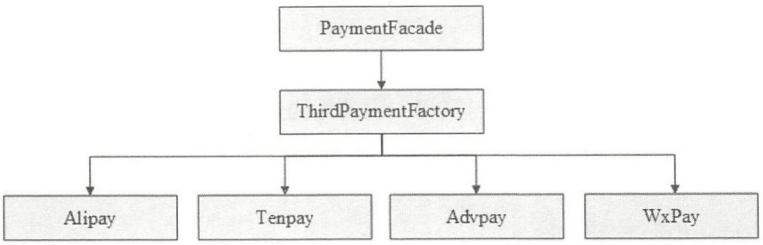

（1）按照公司统一应用分层规范，把第三方支付代理放在 DataLayer 里的同时，每种支付方式都是一个独立的组件，里面的 Model 放在各自组件中，不放在 EntityLayer 层，因为不涉及跨层对象访问。

（2）BusinessLayer 核心类有 xxxLogic、xxxHelper、xxxVerify，如 AlipayLogic、AlipayHelper、AlipayVerify，采用统一的接口编写。

（3）Payment.Services 可访问部分外网，但不可被外网访问，Payment.Notify 则允许被部分外网访问。尽可能地限定网络层访问、设置域名或 IP 安全，在部署时也可用金融级硬件或集群，以提高支付网关的可用性。

### 3. 业务核心代码

业务模块如下图所示。




（1）PaymentFacade：提供对外访问的“门面”。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（2）ThirdPaymentFactory：根据请求的支付类型，创建相应的支付业务逻辑处理类。

（3）关键点：面向接口编程。

```
public class ThirdPaymentFactory{
    public static IPaymentService Create(PayChannels channels)
    {
        if (channels == PayChannels.Alipay)
        {
            return new ThirdPayment.AlipayLogic();
        }
        if (channels == PayChannels.Tenpay)
        {
            return new ThirdPayment.TenpayLogic();
        }
        //...
        throw new NotImplementedException();
    }
}

public class PaymentFacade : IPaymentService
{
    public TradePayResponse TradePay(TradePayRequest request)
    {
        IPaymentService paymentService = ThirdPaymentFactory.Create (r
equest.payChannels);
        return paymentService.TradePay(request);
    }
    public TradeRefundResponse TradeRefund(TradeRefundRequest request)
    {
        IPaymentService paymentService = ThirdPaymentFactory.Create (r
equest.payChannels);
        return paymentService.TradeRefund(request);
    }
   //…
}
```

### 4. 接口封装情况

我们的各个支付方式接口封装情况具体如下：

（1）支付宝支付接口封装：包含支付、代扣、分润、无密退款、补差、转账、冻结和解冻。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（2）财付通支付接口封装：包含支付、分润、退分润和退款。

（3）预付款支付接口封装：包含支付、分润、退款和余额查询。

（4）微信支付接口封装：包含支付、退款。

## 17.3 统一支付通知

统一支付通知包括同步回调和异步通知。它的流程如下：用户完成支付后，第三方支付平台会分别回调统一支付通知应用系统的同步回调页面和异步通知页面，它收到回调信息后，进行相关记录和处理，然后调用业务应用系统的支付后处理接口或页面。统一支付通知应用系统与第三方支付宝的通知机制有些类似，只是减少了不必要的安全验证。以下探讨 3 个问题：

（1）统一支付通知应用系统是如何通知业务调用方的呢？调用方即业务应用系统在支付时将 ReturnUrl 和 NotifyUrl 传给企业支付网关，这与第三方支付一样，只是业务应用系统的支付后处理仅需处理业务逻辑，无须关心安全验证和支付日志。

（2）统一支付通知应用系统为什么有了同步回调，还需要引入异步通知机制呢？这是为了提高通知到位的可靠性，如果同步通知处理服务失败，那么第三方支付平台的服务器会不断将通知重发给异步通知处理服务，但是重发又不能过于频繁。以支付宝为例：企业支付网关的异步通知处理服务执行完成后必须打印输出 success 字符，否则支付宝服务器会不断重发，直到超过 24 小时，在 24 小时内完成 6 到 10 次通知重发（通知频率：5s、2m、10m、15m、1h、2h、6h、15h）。统一支付通知可以先统一接收第三方的支付通知，然后立即返回成功，最后在内部建立一套统一的重试机制。

（3）回调 ReturnUrl 与通知 NotifyUrl 有什么区别呢？没有大的区别，只是一个有用户界面，一个没有用户界面，它们内部都调用统一的支付后处理服务。

统一第三方支付通知的接入，有利于安全、可靠性及支付运营后台，如统一处理数据签名、提高服务器的物理安全、记录安全审计日志、重试机制、补偿、异常处理等。让业务系统更简单，更专注于自己的业务逻辑。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



## 17.4 Demo 下载

- PaymentDemo 下载地址：https://github.com/das2017/PaymentDemo



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 第5篇

# 进阶篇

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用

# 18 技改之路：从单体应用到微服务

技改是技术改造的简称，是技术的蜕变。本章所谈的技改指的是在公司技术发展的某个瓶颈阶段，按原有的开发和组织方式已经无法“玩下去”，这时公司希望引进架构师或技术牛人来破解当前困局。技改对于公司和技术人员而言都非常难得，参与者多，主导者少。笔者有幸前后主导过 3 次 OTA 系统的技改，规模有大有小，每次技改环境和问题虽不一样，但还是有套路可循的。技改之路少讲技术多讲“路”，我们不过多地关注技术细节和中间件的实现，而重点讲述技改的过程和对技改的思考。

## 18.1 系统背景

### 1. 技术规模

公司：

- 国内领先的 B2B 机票分销平台；
- 实现资本原始积累，财务状况良好。

仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



系统规模：

- 应用为 200+；
- 库为 100+，表为 1 万+。

研发规模：

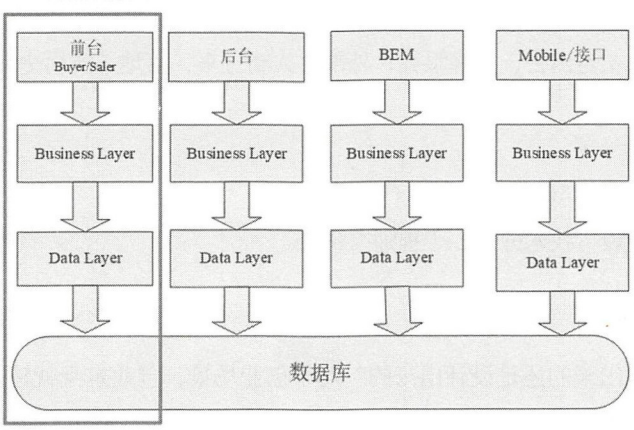
- 开发人员有 200 人左右；
- 服务器有 200 台左右。

此案例中的“主人公”是一个中等规模的电子商务公司。公司从 2006 年的几个研发人员，到技改前的 200 个左右研发人员，业务发展良好，是国内领先的 B2B 机票分销平台。公司之前尝试过 2 次系统重建，请了一批批的牛人，前后经历 4 年技改之路，但都以失败告终。此案例是笔者的第 2 次技改，效果不错，整体进展顺利，团队技术水平也有 1~2 个档次的提升，算是比较成功的实践。

### 2. 单体应用

现有单体应用架构如下图所示。

**4.1 现有架构图**




| 业务逻辑 | 应用 | 重复次数 |
|---|---|---|
| 预订逻辑 | 前台、接口、BEM、Mobile | 4 次 |
| 订单处理逻辑 | 前台、后台、BEM、作业小应用、Mobile | 5 次 |
| 产品供应逻辑 | 前台、后台、BEM、作业小应用、Mobile | 5 次 |
| 财务结算逻辑 | 前台、后台、BEM、作业小应用、Mobile | 5 次 |
| 公共逻辑 | 前台、后台、BEM、作业小应用、Mobile | 5 次 |



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 3. 主要问题

- 单体应用，该合并的没有合并，该拆分的没有拆分，单个体量不合理，主平台体量太大，其他又过小。
- 技术过旧：使用 7 年以前的技术，主平台因采用单体式架构，且体量过大，无法整体更新维护。
- 多版本共存：版本混乱，只敢添加，不敢修改。
- 整个系统非常脆弱，问题多，访问量一大就“挂”。
- 管理问题：发布困难，测试困难，修改困难，排错困难。

## 18.2 前期工作

### 1. 架构部组建

成立架构部：

- 内招几名老程序员，外招几个架构师。

培养：

- 内部人员走出去，提高眼界；外部牛人请进来，落地了解历史和业务。

制度：

- 项目管理+知识分享：JIRA+Wiki；
- 团队建设，技术分享，工程师文化。

### 2. 总体规划

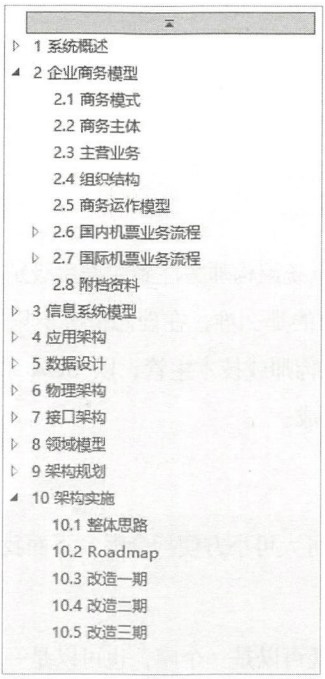
架构是演化出来的还是设计出来的？对于创业场景，创业本身就是在未知中寻找机会，将不清楚变为清楚，系统的架构自然是演化出来的。而对于技术改造或 Google 搜索等复杂工程场景，系统的架构当然要精心策划。我们整整花了 1 个多月的时间，对公司电商系统总体架构做了规划，然后对内宣讲推广，让每一位参与者了解自己的目标和价值。不手握地图，你怎知站对了位置？总体规划文档如下图所示。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用






### 3. 中间件构建

我们构建的中间件有：

Job/Redis/CenterLog/业务监控（Metrics）/Dashboard/调试工具（WinDbg）/RabbitMQ/ORM 工具（Dapper）/MongoDB/JEtermClient/公共类库（JFx）/ZooKeeper/OpenTSDB/HBase/Search 工具（Solr）/元数据管理（DDM）/DLL 管理（NuGet）/自动发布（Jenkins）/微服务架构（JSOA）/

中间件是应用系统的基础设施，是应用的装备和工具。以上中间件的构建过程贯穿整个技改的生命周期，每一个中间件的构建可能需要 1～2 个月，它们大部分都基于开源。请关注上面的顺序，直面当前的问题，按需快速构建和推动。虽然使用开源软件，但中间件的引进和改造有自己的一套流程：调查→试用→选型→深入研究→Demo→Wiki→分享推广→业务系统试用→改进完善→大规模推广。中间件的构建和增加，不仅对当前业务系统影响较小，还可以解决一部分业务难题，减轻数据库的压力。同时它还有利于建立技术氛围和分享机制。一个有激情、爱技术的研发团队，对技改的具体



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



实施是非常重要的。

## 18.3 技改实施

### 1. 数据库改造

当面对 100 多个库时，系统架构师关注到数据库级别即可，建库拆库。数据库按模块整体迁移，其实并没有想象的那么难，在理想情况下只需修改数据库的连接，而对于表和字段的优化，可由应用架构师或技术主管，以 SOA 收口或应用重构的方式来实现。数据库改造和设计由三部分组成。

（1）数据库伸缩。

数据库如何做到可伸缩，可大可小方便拆分呢？下面我们结合 2.2 节那张 E-R 图来说明：

① 从内往外看，一个框既可以是一个库，也可以是一个模块，还可以是一个表，根据当前业务规模和系统复杂度来实现。

② 我们的大实体关系图具体为：产品、用户、订单、结算、基础设施。它们早期可以是一个库，里面有 5 个模块，中期可以分为 5 个库，后期则可以以更低级别分为更多的库。

③ 命名规范。数据库名：业务线缩写+库名+DB；模块名：参考大 E-R 图+专业词汇缩写；表名：模块缩写+表名；自增编号：表名+ID。

④ 模块内可多表连接，模块间减少连接，数据库间不允许连接。

⑤ 每一个数据库有且仅有一个 Owner 组，原则上只允许一个团队才能“Create”，其他团队访问需要分级控制，L1 为接口，L2 为只读库，L3 为直接读写“写库”。

（2）数据库规划。

数据库规划如下图所示。



仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



### 9.4 数据库规划

| 数据库 | 说明 | 备注 |
| --- | --- | --- |
| FltProductDB | 国内机票产品库 | 包括政策和基础价格 |
| FltOrderDB | 国内机票订单库 | |
| FltCommDB | 国内机票公共库 | |
| FltAccountDB | 国内机票结算库 | |
| FltProductLogDB | 国内机票产品日志库 | |
| FltOrderLogDB | 国内机票订单日志库 | |
| FltAccountLogDB | 国内机票结算日志库 | |
| | | |
| IFltProductDB | 国际机票产品库 | 包括政策和基础价格 |
| IFltOrderDB | 国际机票订单库 | |
| IFltCommDB | 国际机票公共库 | |
| IFltAccountDB | 国际机票结算库 | |
| IFltProductLogDB | 国际机票产品日志库 | |
| IFltOrderLogDB | 国际机票订单日志库 | |
| IFltAccountLogDB | 国际机票接口日志库 | |
| | | |
| UserDB | 用户库 | 将来合成为一个库，现在有国内、国际两个库 |
| EmpDB | 雇员库 | 同上 |
| MDMDB(ip\city\mobile\airLine) | 基础主数据 | 同上 |
| CMSDB | 网站内容管理库 | 同上 |
| CRMDB | 客户关系管理库 | 同上 |
| PaymentDB | 支付库 | 同上 |
| PaymentLogDB | 支付日志库 | 同上 |
| InsuranceDB | 保险库 | 同上 |
| | | |
| FxDB(mq\cache\job\config) | 框架库 | |

数据库是整个信息系统中生命周期最长、最难修改的部分，所以让时间来解决时间的问题，要加强设计，具体实施过程如下：

① 在推广地图即总体架构文档后，我们就新建立了一批库，这在早期还遭到 DBA 的抱怨。

② 新增相关库后，新表按新规则创建，特殊情况走特殊审批流程。

③ 去 SP、去关联，让数据库服务器减少业务逻辑计算，专注于存储功能。

④ 数据库拆分，改表改字段，采用模块整体迁移或应用重构方式。

⑤ 一年后再去看数据库，发现在没有特别立项和驱动的情况下，已接近一半的表在新库中。



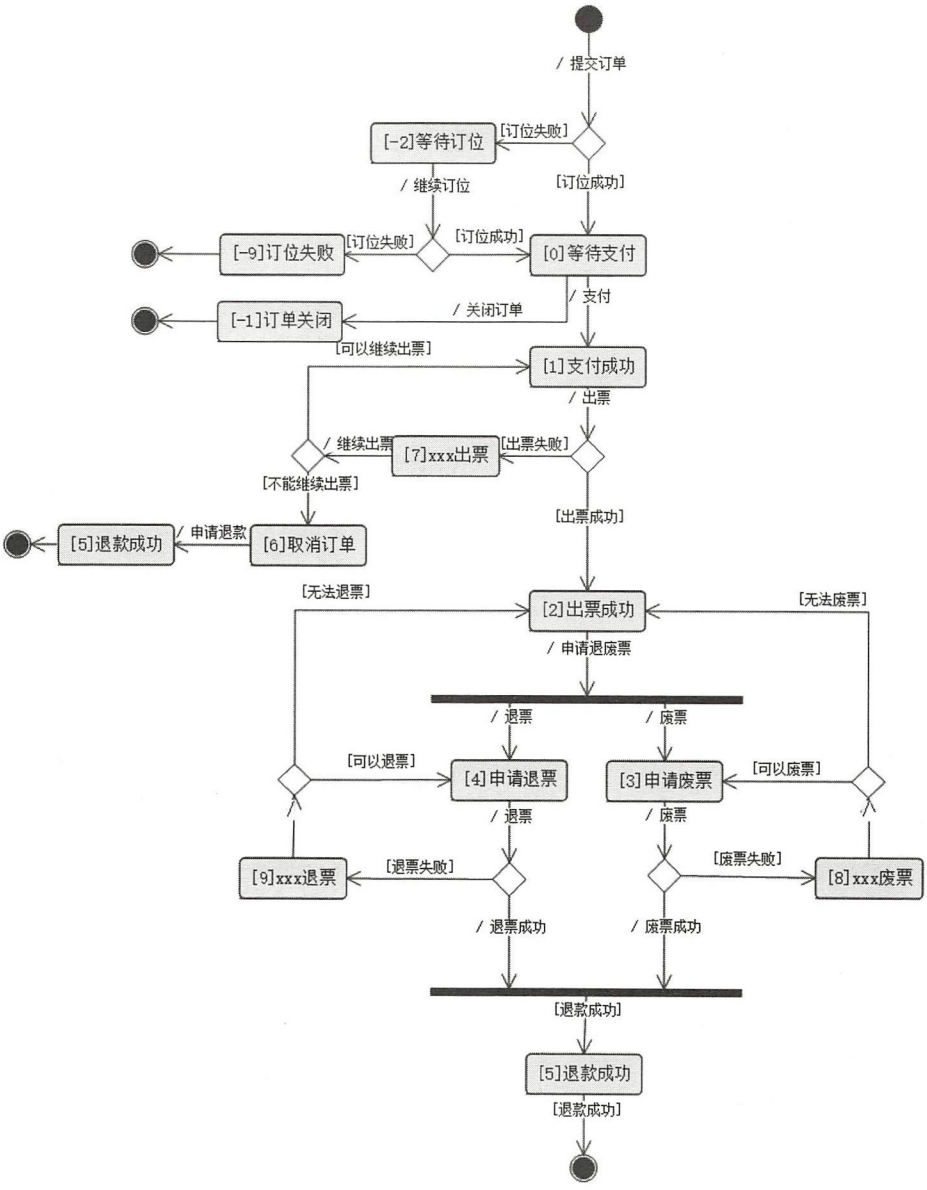
仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



（3）数据变迁。

状态图表示的是数据的变迁，是数据与行为的互动，数据的变化会引起行为的变化，行为的变化会产生数据的不同。下图是国内的订单状态变迁图，它的价值不仅体现在数据库层，还体现在 SOA 服务化和核心业务流程。






仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



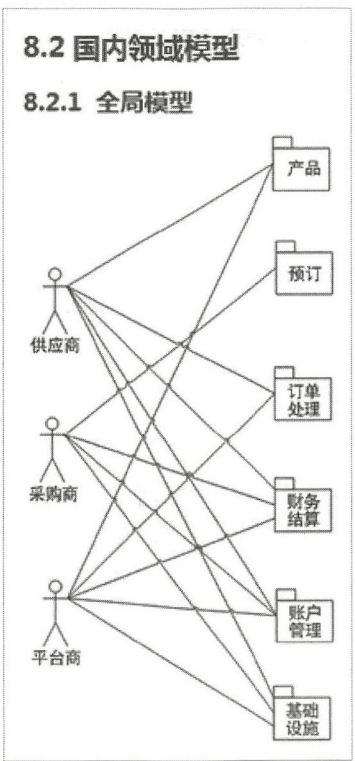
### 2. 服务改造

服务是动词，是行为或活动的抽象，它的价值在于业务逻辑或行为的重用，具体实施过程如下：

- 服务列表和服务协议，在设计阶段使用 Excel 表格。
- 统一 Request/Response 规范。
- 服务实现。因为没有直接可见的业务价值输出，所以最好以工单或项目来落地。
- 服务治理。早期没有工具时，使用 Wiki 做简单管理，后期使用专业的服务治理工具。

（1）领域模型。

全局模型如下图所示。




我们画领域图的架构师是 2 位老员工，他们之前没有画过 UML，但我们的状态图和

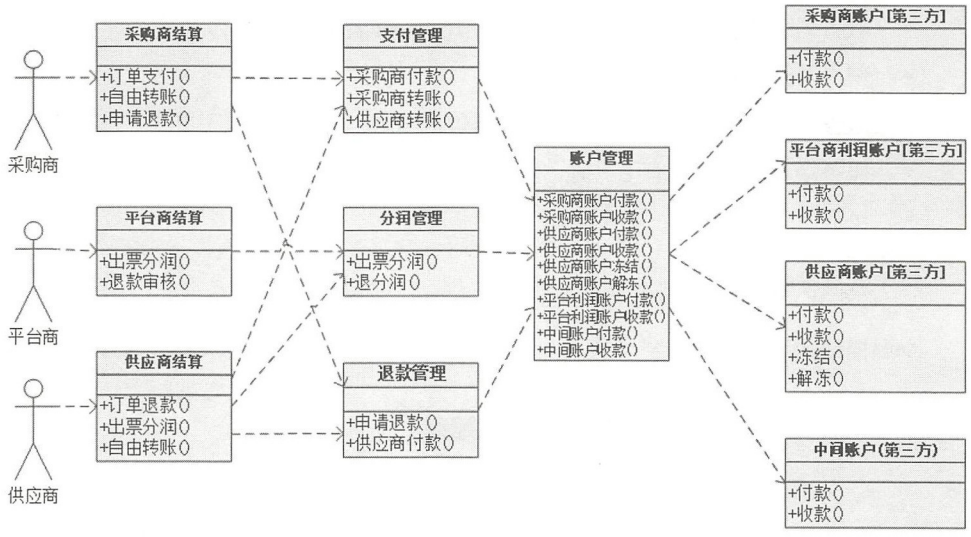


仅供非商业用途或交流学习使用

非卖品，仅供非商业用途或交流学习使用



领域图都是出自他们之手。其实画领域图的关键是懂事物本身，并知道它们的关系。我们的领域图与业务模型中的五大业务流程一一对应，包括预订流程、订单处理流程、产品供应流程、财务结算流程和账户管理流程。模型图如下图所示。




（2）微服务。

我们的微服务 JSOA V2.0 是基于 ServiceStack 当时最新的版本号 4.0.50 实现的，它本身支持轻量级协议和 Metadata，以及 Swagger，是微服务的一种架构实现。另外，它还可以以 API Gateway 的方式实现 Open API。

### 3. 应用架构改造

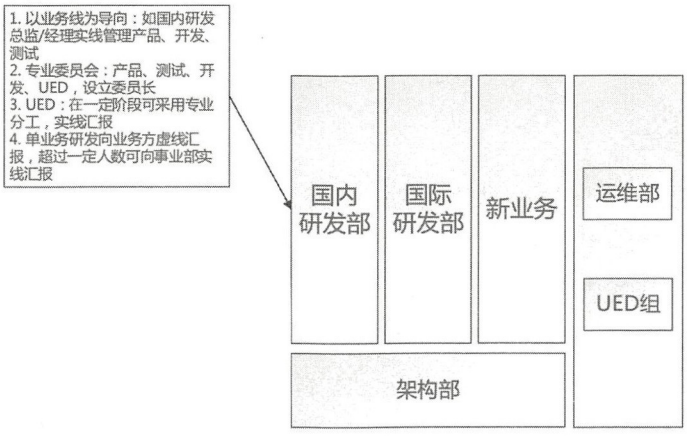
系统是什么？系统=元素+关系。应用架构是什么？应用架构=应用+架构。应用就是系统的最小单元，应用分级和应用编号则构成了应用关系即应用的架构，它有利于应用的管理、交互和追踪。应用分为产品线、子系统和应用三级，每一级编号为 2 位数，如 100206。应用要从用户的视角出发，先有用户，然后有应用功能，这样才是以用户为中心去构建系统。

### 4. 组织架构微调

组织架构如下图所示。

非卖品，仅供非商业用途或交流学习使用






组织架构没有最佳实践，只有适合于自己当前的选择，以下是对组织架构与技术架构对齐方面的思考。

（1）艺术与工程相分离：UED。

（2）软件开发与硬件相分离：运维。

（3）技术研发与业务研发相分离：架构部。

（4）需求、实施、验收相分离：每条业务线分产品组、开发组、测试组。

（5）开发按业务职责相分离：预订组、产品组、订单组。

（6）采用专业技术委员会制：测试、产品、开发轮流主持，设委员长。

## 18.4 总结

### 1. 过程总结

- 第一步总体规划：手握地图，明确路线。
- 第二步数据库：建库拆库，去 join 连接，去 SP 存储过程。
- 第三步中间件：按需构建，先增加常用中间件。
- 第四步服务：技改=工单，有业务价值输出。
- 第五步应用：拆应用，建门户 Portal，重构应用。



非卖品，仅供非商业用途或交流学习使用



- 第六步组织架构微调：技术架构与组织架构对齐，技改之后调整。
- 第七步固化：框架化、自动化、管理过程工具化，如 DevOps。

### 2. 经验感悟

- 从服务入手是错的，从数据库或中间件入手是正确的。服务属于高级阶段，方便行为的重用，是深层次优化，但太慢了。
- 从当前问题或故障入手，要先“灭火”，逆向分析 Dump 工具很重要。
- 历史要尊重，早期不可做大的改动，不能过多地影响现有业务。建议只做加法，建新库和新中间件，这样就不会有太多阻力和负担。
- 一般不能全部重建，除非系统较小，当系统规模大时只能拆分后分步重构。
- 技术并不是技改过程中最复杂的，人和事及关系才是麻烦的部分，历史问题的背后是人。
- 每次的环境和问题都不一样，要有准备脱一层皮的心态。

### 3. 通盘无妙招

技改是大折腾，于公司于个人而言都是“小改怡情、大改伤身”，我们应该避免大的技术改造，但此现象又比较常见，特别是业务发展快的创业公司。所以真正的高手下棋，应该是通盘无妙招，让正确的事情很容易发生，基于自然的演化来实现技术的演进。怎样才能通盘无妙招，实现系统的良性长久发展？我们需要两个力量，一个是技术，一个是业务，如果只重视业务，则很容易在技术上“积劳成疾”；如果完全是技术驱动，则又容易忘记业务目标。所以它们应该相伴相生，共同发展，在大的技术改造实施之后，在框架和流程相对固化后，小的技术重构项目应该长期存在，这样才能实现良性循环，让系统进入自然演进的状态。

## 18.5 互动问答

**问题：请问张老师，如果再来一次技改你会怎么做？在你做过的技改过程中你觉得最大的收获是什么？觉得做得不好的又是什么？**

这个问题非常好，为了更好回答您的问题，我简单介绍一下本人的 3 次技改经历。我的第 1 次技改是重建，项目从第一年 10 月份到第二年 8 月，历时 10 个月，可以说是



非卖品，仅供非商业用途或交流学习使用



技术成功、项目失败。第 2 次技改是重构，只管技术，少管人和业务，整体效果好，可以归结为成功。第 3 次技改还是重构，既管技术，又管人，且业务处于高速发展期，资源少，可以总结为技术与业务相伴相生，技术效果一般。如果以后还有机会，自己就不再直接负责了，实在是太累，从内外部各招几个架构师，并按上面的工作流程和方式，把握好技术与业务的关系和资源占用即可。回到具体问题：

（1）如果再来一次，我会多参考第 2 次技改经验，即此章分享的过程总结。

（2）技改后的收获是“脱胎换骨”般的，技术和管理都有很大提升。

（3）不好的地方：要更多地关注业务，以及平衡好业务与技术的关系。

**问题：单体应用向微服务迁移时，平滑过渡有什么技巧？如何解决分布式事务的一致性呢？还有关于微服务持续交付、测试、监控（语义监控）方面有落地工具吗？**

（1）平滑过渡的技术：直面当前系统的问题，不断有价值输出，然后参考上面的过程总结，先是规划，然后是中间件和数据库，最后是服务和应用。

（2）分布式事务的一致性：使用替代方案，如最终一致。

（3）落地工具：MSA 与 SOA 的治理没有本质差别，还是 DevOps/Trace/Metrics/，我们使用的是 ServiceStack/Metrics/CenterLog/Jenkins/。

**问题：如果旧的模块关联复杂，又影响现有系统性能；相关开发人员流失；不好梳理，改造有风险；重写老板不答应。则该如何取舍呢？**

（1）旧的模块关联复杂，又影响现有系统性能：先“灭火”解决当前故障，逆向分析 Dump 工具。

（2）相关开发人员流失：引进中间件，建立分享机制和学习型团队，讲技改总体规划，让每个人了解自己的价值和目标。

（3）不好梳理，改造有风险：内招几位老员工成为应用架构师。

（4）重写老板不答应：有业务价值输入，技改等于工单或项目，借助业务项目来实现技改。



非卖品，仅供非商业用途或交流学习使用

# 19 机票垂直搜索引擎之性能优化

## 19.1 行业背景与垂直搜索

从 2011 年到 2016 年，无论国内，还是国际，整体趋势都是机票价格便宜了，坐飞机的人也越来越多。特别是国际机票，这五年里机票价格下降了 30%，客运量增长了 140%。

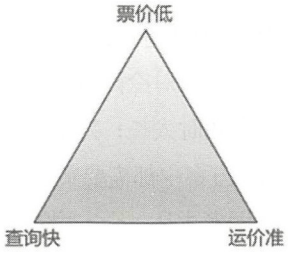
乘客越来越多，购买机票的渠道有哪些呢？现在主要有三个：网络平台、代售点和航司官网。像携程、去哪儿、飞猪、同程等，都是主流的网络购票平台；像旅行社这类代售点，是旅行团的主要购票渠道；同时大部分航空公司的官网也可以购票，而且价格相对较低。总体来说，网络平台是最大的销售渠道，占比为 76%。为什么网络平台占有这么大的份额呢？主要原因是机票垂直搜索引擎是主要的用户流量入口，用户一般是先比价，然后去预订，一个好的机票搜索引擎查询的产品丰富、价格便宜，而且响应速度快，运价也准，这些特性在技术方面实现好并不容易。

非卖品，仅供非商业用途或交流学习使用



## 19.2 主要问题与解决方案

机票查询要快、准、低。快是指查询快，能够提供一个良好的用户体验；准是指运价准，可以保证出票的成功率；低是指票价低，能够吸引更多的用户。如果票价要有优势，就要有大量产品，产品数据多了查询就慢；如果查询要快，就必须要有缓存；数据缓存了，运价就可能不准。这三者是矛盾的，类似于 CAP 原则，具体示意图如下。




对于以上问题，怎么解决呢？通用的三个技术方案有：用 DB+Redis 平衡响应速度、数据实时性和查询成本；用削峰填谷的 MQ 来处理高并发；将业务服务化、模块解耦。这些只是通用的技术点，并没有什么难度，我们这里重点介绍与最终结果密切相关的四个模块：静态数据、缓存策略、实时查询和政策匹配。

（1）静态数据：能静态处理的数据尽量静态化，存储到本地，可以是数据库或缓存，以方便快速地查询，如航班信息、运价数据和政策数据等。

（2）缓存策略：从中航信拿到运价数据之后，进行热冷门数据分类，数据永不过期但持续更新，自主控制数据的更新频率。

（3）实时查询：多渠道多供应实时获取远端数据，多数据源查询速度会变慢，远端服务不可控。解决方案是三段超时，即前端用户超时、中端运营超时和后端供应超时。

（4）政策匹配：大量的产品数据和大量的业务规则不可能都提供给用户，需要通过一定的算法进行匹配过滤、排序等。

## 19.3 静态数据与任务打底

机票查询的静态数据主要有城市、机型、航司、运价数据等，这里重点介绍较为



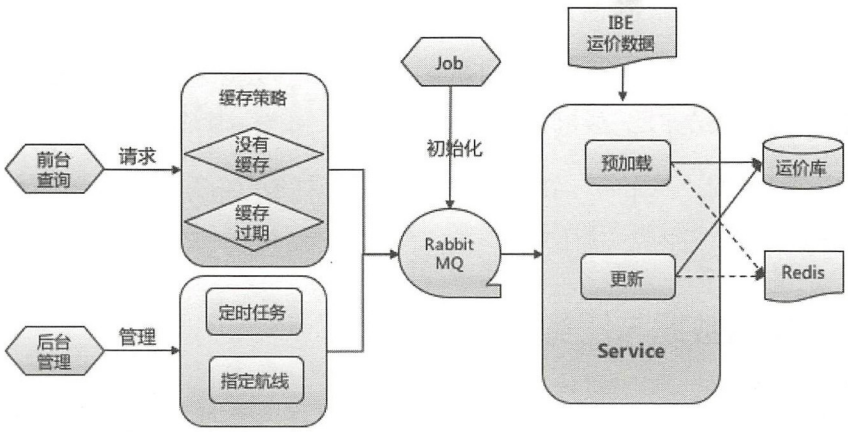
非卖品，仅供非商业用途或交流学习使用



复杂的运价数据。运价数据的获取虽然间隔时间较长，但数据量大且更新频次不同。运价数据是由中航信统一提供的，有两种途径：黑屏查询和 IBE 接口，将获取的数据保存到数据库和缓存中，用户查询的时候直接从缓存中获取，同时会按照一定的缓存策略来更新。

最初我们设计了两套方案来打底运价数据，两个方案各有优劣。方案 1 是先预加载所有的运价数据，然后全部保存到数据库和缓存中，在航班查询时通过缓存策略进行相应地更新；方案 2 是把运价数据根据航线查询频率分为热门和冷门数据，然后每天凌晨对热门数据预加载，并在航班查询的时候对冷门数据进行更新。可以看出，方案 1 能保证数据的完整性和实时性，但预加载用时太长；方案 2 能控制预加载用时，但热门数据的实时性会从早到晚逐渐降低。两个方案中都需要实时更新，在考虑数据实时性的同时，还要考虑获取数据的费用，平衡好两者才是一个实用的方案。

综合对比之后，我们采用了方案 1，具体实现如下图所示。首先通过 Job 对运价数据进行初始化，然后以任务消息的方式发送给 MQ，MQ 里的消息会被后台服务自动消费，执行消息队列里的任务，把运价数据保存到数据库和缓存中。数据预加载之后，用户在前台查询时，如果缓存里面没有数据，或者查到的缓存数据是过期的，那么系统会自动发一条任务消息给 MQ，或者人工配置指定的航线定时更新，Job 也会自动发送任务消息给 MQ，前台和后台的消息被服务消费以实现数据的更新。用户的不断请求和后台指定的任务保证了数据的持续更新，时间越久，数据的准确性越高，用户查询的命中率也会越来越高。



非卖品，仅供非商业用途或交流学习使用

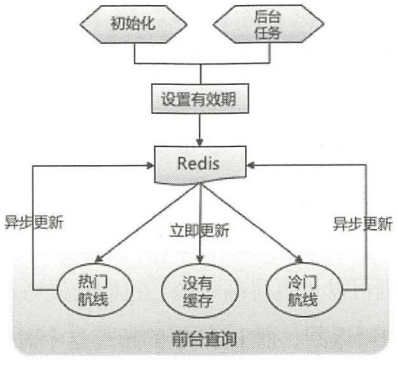


## 19.4 缓存策略与数据一致

上面说到运价数据同时存储在数据库和缓存中，为什么有了缓存还要数据库呢？存储到数据库是为了方便数据的多维查询和管理，包括对缓存的进一步干预。数据库查询的功能强大，但速度慢，缓存的性能好，但从缓存里获取的数据会有不准确的问题。怎么才能做到查询快而且数据准呢？我们的解决方法是缓存永不失效、数据分类、自主控制更新频率，以实现运价数据的又快又准。

根据航线查询的频率，将可以分成热门数据、冷门数据和没有数据，航班多、查询多的是热门数据，航班少、查询少的是冷门数据，查询不到就是没有数据。在预加载或更新运价数据时，将缓存设置为一个较长时间或永不过期，然后在前台访问时，不同数据类型采用不同的更新策略，具体如下图所示。

- 热门航线查询，在缓存中获取数据，数据中有一个自己的缓存时间字段，然后根据这个时间来分别进行处理。
  - 1 小时之内更新的：新鲜度较高，可以直接用；
  - 1-6 小时之内更新的：预警 n 次，第 n+1 次命中时则异步更新运价；
  - 6 小时之外更新的：新鲜度太低，异步更新运价。
- 冷门航线查询与热门航线一样，只是不预加载且缓存时间稍长。
  - 12 个小时之内更新的：新鲜度较高，可以直接用；
  - 12-48 个小时之内更新的：预警 n 次，第 n+1 次命中时则异步更新运价；
  - 48 个小时之外更新的：新鲜度太低，异步更新运价；
- 缓存没有数据时，直接获取最新运价，同时更新数据库和缓存。



非卖品，仅供非商业用途或交流学习使用



无论预警后更新，还是直接更新，都是先把缓存中的数据返回给用户，同时异步更新数据库和缓存。虽然存在数据查询不准确的概率，但被用户再次查询时就准确了。查询到的数据即便不准确，在后继的航班预订时也会进行二次的验舱验价，运价数据和库存数据会再次更新。用户不断地查询，数据不断地更新，查询命中率就会越来越高，并且用得人越多情况会越好，会逐步趋近于 n 个 9。

## 19.5 实时查询与三段超时

能静态化的数据要尽量静态化，但远端数据的实时查询还是必不可少的。实时查询如何做到又快又好呢？特别是多数据源、多供应商的实时查询场景。我们的国际机票查询就是这样的，前台页面点击查询时实时调用供应商接口，早期我们仅调用一个供应接口，产品比较单一，数据不够丰富。后面引入了多供应商，产品变丰富了，也有了低价、但同时带来了很多新问题，比如供应端接口需要 20~30 秒，但前端客户只能接受 8 秒以内，怎么办？提高供应数据门槛？但这不是核心竞争。还有查询速度变慢、外部数据源不可控、数据格式多样等问题。

对于以上问题，我们的解决办法是三段超时，所谓三段，即供应端、运营端和客户端。前端满足客人、中间满足运营控制策略、后端满足供应商，三方都要满意，这样才能使产品更丰富、价格更低、运营策略更灵活、用户响应更及时。三段超时的时间可以根据具体场景进行配置，具体如下图所示。

- **供应端超时：**供应端是后端，是指提供数据源的一方，供应端存在的问题就是外部不可控。供应端处于数据来源的底端，解决办法是尽量加大供应端的超时时间限制。我们对请求供应接口的最大 HTTP 超时时间设置为 45 秒，这个值可以满足绝大部分情况。
- **运营端超时：**运营端是中间端，获取供应商的数据之后，做包装转换、去重、政策匹配等业务处理。我们先统计每一个供应接口的请求时间，确认供应接口数据的质量和优先级。比如 A 供应数据的质量相比 B 和 C 供应数据的质量要高，那么 A 的请求级别可以设置得高一些。我们优先考虑获取 A 供应的数据，如果 A 的数据在 8 秒内就返回，而 B 和 C 的超过这个时间，那么此时在前台就只把

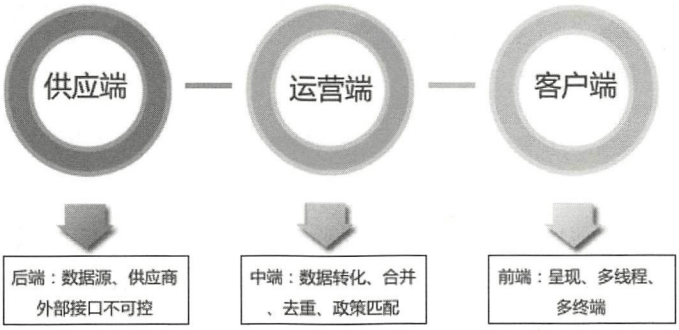


非卖品，仅供非商业用途或交流学习使用



A 的数据返回给客户。对于 B 和 C 的数据，由于在 HTTP 请求时我们采用异步并设置了较大的供应端超时，所以它会在 A 返回之后，继续异步请求并将返回的数据保存到缓存中，以供用户下次或其他用户使用。当我们获取了多供应商的产品数据后，这时会有一定重复的数据，需要进行规范化处理，将不同数据格式转换成统一标准，然后去重并选取最优，最后根据运营策略进行政策匹配等。

- **客户端超时：**客户端是前端，需要处理最终展示和不同终端用户的不同需求。客户端采用多线程异步读取，这样不会影响主线程的速度，同时并发请求，提升响应速度和用户体验。这里的主线程请求时间可以理解为前台终端设备需要等待的时间，比如 APP 要求 8 秒内返回，那就设置为 8 秒；如果 PC 端 B2B 白屏网页查询，客户可以等待时间为 25 秒，那么就设置为 25 秒。客户端的超时时间要大于或等于所有的运营端超时时间，例如客户端超时是 25 秒，那么运营端线程 A 的超时时间最大可以为 25 秒，如果线程 A 的绝大部分航线获取时间是 18 秒，那么线程 B 和 C 的超时时间最好不要超过 18 秒，这里的用户体验要综合考虑概率问题。




## 19.6 政策匹配与算法优化

弄来这么多产品，不可能都提供给客人，需要根据运营规则来进行匹配。机票政策就是机票产品的运营控制策略，如下图所示，包括政策类型、客户类型、航程类型、乘客类型、航司、航班、舱位、城市、日期、返点 、定额、Office 号等多种属性。

非卖品，仅供非商业用途或交流学习使用




为什么有这么多属性呢？因为机票产品的运营规则很复杂，而这种规则的复杂性，直接导致在航班查询的时候，机票政策的匹配也很复杂。对于这种大数据、复杂业务规则的数据处理，需要有一套专门的政策匹配算法，具体如下：

第一步是直接从数据库查政策，在前端查询的时候，根据查询的条件，如出发到达城市、日期等，从数据库中大范围地获取政策数据，并把这些数据放到内存中。第二步是在内存中对每个产品进行政策匹配即过滤，先将每一个属性转化为业务规则如限制城市、排除供应商、航司指定供应商等，一个属性一个类，采用统一的接口，然后增加到政策过滤器中。产品与政策的匹配过程就像水流过过滤网一样，把最优政策应用到产品



非卖品，仅供非商业用途或交流学习使用



上如调整价格。这个过程有些复杂，为此我们编写了一套自己的政策过滤器 PolicyFilter 框架。第三步是按照政策返点高低进行排序。第四步是将最优政策返回给前台。以下是部分核心代码的演示：

```
        /// <summary>
        /// 获取针对主平台的过滤器
        /// </summary>
        /// <returns></returns>
        2 个引用
        private static GetPolicyFilterCore GetInstanceForMainland()
        {
            GetPolicyFilterCore sfc = new GetPolicyFilterCore();

            //强制排除供应商
            sfc.AddFilter(new ExcludeRatePidFilter());
            //供应商上下班时间
            sfc.AddFilter(new ProviderWorkTimeFilter()); //全局1
            //限制专供政策
            sfc.AddFilter(new ShowTypeUserForMainLandFilter());//全局
            //限制城市
            sfc.AddFilter(new RatePolicyTypeFilter());//全局
            //区域限制
            sfc.AddFilter(new RateAreaAgentFilter());//
            //特殊政策指定供应商
            sfc.AddFilter(new SpecialRateFilter());//2

            //航司指定供应商政策
            sfc.AddFilter(new AirComProviderFilter());
            //灰名单用户航司指定供应商
            sfc.AddFilter(new AirComWithProviderFilter());
            sfc.AddFilter(new AirComWithBlackUserFilter());
            sfc.AddFilter(new AppointProviderByUserForMainlandFilter());
            return sfc;
        }
```

```
                //政策过滤 【挑选出有效的政策，不修改政策值，仅过滤】
                //航班查询的时候 isNeedFilterPolicy = false
                if (isNeedFilterPolicy)
                {
                    int amount = request.Amount;
                    request.Amount = policyList.Count;

                    //政策过滤 过滤器只会对政策进行取舍 不会修改政策的属性值
                    policyList = FilterPolicyList(request, policyList);

                    //对政策进行贴点高反及排序
                    policyList = SetPolicyDiscount(request, policyList);

                    //设置默认政策
                    policyList = SetDefaultPolicy(request, policyList);

                    //去掉同一供应政策
                    policyList = DistinctPolicy(policyList, amount);

                    //设置退改签信息
                    policyList = SetRefundInfo(request, policyList);

                    //初始化供应出票速度、评分、手续费
                    SetProviderInfo(request, policyList);

                    //政策排序
                    policyList = SortPolicy(request, policyList);

                    //保底政策指定供应商的，则在相同返点保底政策项中优先显示
                    ProviderForOtherPolicy(policyList);
                }
            }
            //婴儿政策
```



非卖品，仅供非商业用途或交流学习使用



## 19.7 小结

机票垂直搜索性能优化不仅适合于机票行业，也适合于其他垂直行业，在垂直搜索引擎方面有一定的通用性，只要它存在：远端数据获取、静态数据、缓存更新、规则匹配、多数据源等问题，都是类似的解决方案。垂直搜索主要有四把“刷子”。第一把刷子是静态数据与任务打底。第二把刷子是缓存与更新，保持数据的新鲜度，不仅要快，还要准。第三把刷子是实时查询与三段超时，多供应商多数据源，供应商要 20 秒，客户只能接受 3 秒，怎么办？解决办法是三段超时。第四刷子是政策匹配，好不容易弄来这么多产品，不可能都直接显示给客人，需要根据运营规则进行匹配。每一个具体的技术可能并不复杂，但把它们综合起来，解决具体的实际问题，为公司为行业带来价值，并不是一件容易的事。技术的核心价值在于技术的应用，技术价值要借助技术应用和产品才能发挥出来，这比单纯的技术学习要有意思得多，希望以上能应用到你具体的工作中。

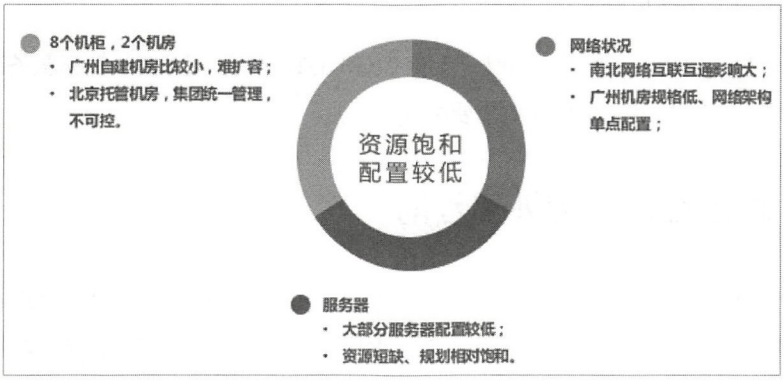


非卖品，仅供非商业用途或交流学习使用

# 20 上云纪要

## 20.1 为什么要上云

在我们部署到公用云之前，公司的资源状况、平台状况及遇到的问题如下图所示。




- 资源状况

  我们共有 8 个机柜和 2 个机房。广州自建机房规格低、难扩容，北京托管机房由集团统一管理，不可控，而且大部分服务器配置较低、资源短缺、规划相对饱和，网络也存在南北互联互通问题。

非卖品，仅供非商业用途或交流学习使用

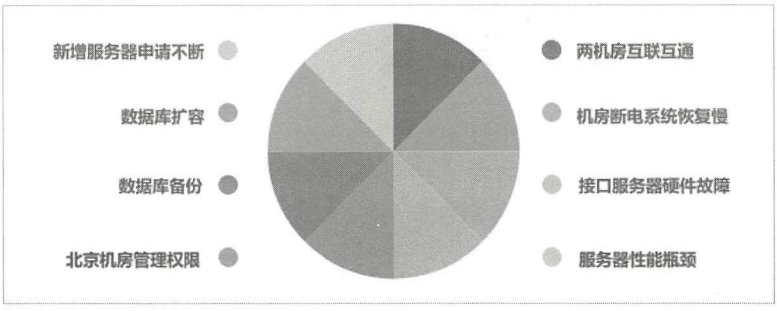


- 平台状况

  应用架构不合理，单点严重，随时可能发生严重故障。

- 遇到的问题

  这样的架构自然会爆发一系列问题，以下是最近发生的八个问题，如下图所示，均无法快速、及时地解决，这里重点介绍其中的四个问题：




（1）新服务器申请频繁，比如我们在上云前一个月申请了 24 台服务器。

（2）北京机房没有完整的管理员权限，由集团统一管控，故障处理困难。

（3）网络互联互通问题和机房断电问题，例如在 2017 年四月份业务高峰期断电 1 小时，导致 450 万资金的损失。

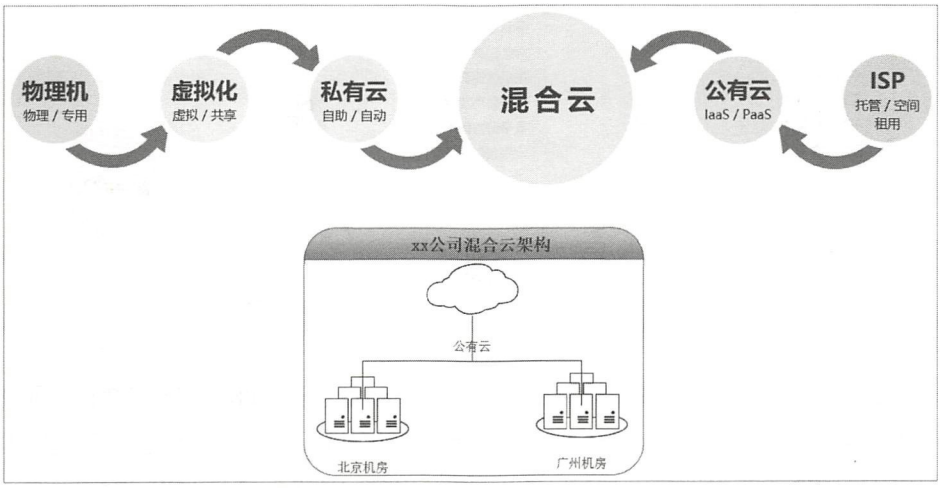
（4）数据库备份扩容问题，当时一天有 21GB 的数据库增量备份，扩充服务器和机柜需要集团层层审批，整个流程大约需要两到三个月。

## 20.2 内部虚拟化和外部云化

以上问题都无法快速及时地解决，怎么办呢？我们的解决思路是“内部虚拟化，外部云化”，即内部的服务器增加硬件，通过虚拟化技术升级为私有云，然后将北京托管机柜逐步迁移到公有云，最后用 VPN 把北京机房和广州机房分别与公有云打通，以公有云为中心组合成混合云，形成一个大的局域网，如下图所示。

非卖品，仅供非商业用途或交流学习使用





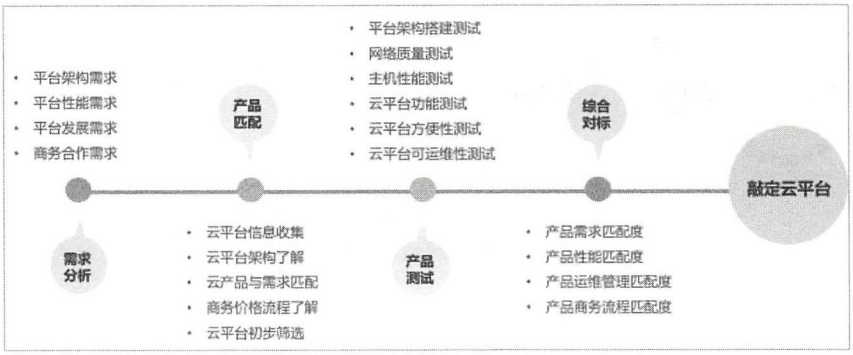

这个事情我做不了，需要找一个靠谱的人来执行。我希望这个人拥有主流云的运维经验、电商运维经验和虚拟化经验，当然还要拥抱新想法，思维要一致。此人可以外招，也可以将内部人员送出去培训，例如参加一些外部的技术会议。我们选择的是前者，因为我们的运维人员当时已经习惯了自建机房和部署物理机的方式，短时间内改变的成本较大。

## 20.3 云选型

确定了解决思路，接下来就是云平台选型了。国内主流云平台众多，该如何选择呢？首先是需求分析，深入了解我们的技术和业务的各项需求，包括平台架构需求、平台性能需求、平台发展需求、商务合作需求。比如我们有 3 个项目都需要加密狗服务，而一些云厂商并不能很好地支持，但这却是我们的一个硬性需求。其次是云产品分析，即将云平台的产品拆开进行单项产品比较，包括主机性能、网络质量、云平台的易用性和可运维性等。然后是与云厂商沟通交流，云平台的试用和测试。最后结合自身需求进行综合选择。过程如下图所示。

非卖品，仅供非商业用途或交流学习使用






经过初步选型后，我们最终回到了大公司与小公司的终极 PK 问题。这时公司内部有两种声音，第一种声音是选择知名品牌如阿里云或腾讯云，第二种是选择满足我们当前需求且可定制的云提供商如 xx 云。不太了解云计算的技术人员大部分会选择阿里云或腾讯云，因为它们是大企业，这与我们以前组装计算机一样，不懂的人会选择品牌机，而我们的运维团队则更倾向 xx 云，认为 xx 云的混合云定制和服务更好。经过大家共同的决策，我们最终选择了 xx 云，主要有以下五个原因：

（1）加密狗，我们的应用软件小太阳、泰比 OCR、胜意，都需要加密狗服务。

（2）专线接入，方便对接北京机房形成混合云。

（3）简单易操作，用户体验好，大家都喜欢 xx 云界面的小清新风格。

（4）用户服务好，一对一跟进的企业式服务。

（5）标准开源的技术服务，没有二次包装，让用户更放心。同时有利于私有云和公有云的统一管理。

## 20.4 上云八条

确定云平台后，接下来的工作就是迁移上云。具体如何迁移上去呢？我们一起制定了一个指导性的建议，即上云八条，前四条为网络架构。

（1）20 倍规划、5 倍设计和 1.5 倍实施。规划和设计要大一些，但实施时小一些，这样不仅便于将来的扩展，也节省了当前的费用。

非卖品，仅供非商业用途或交流学习使用



（2）两个逻辑网络：一个内网和一个外网，两个负载均衡，两个防火墙，安全隔离内外网。

（3）五条产品线：国内、国际、商旅、旅游及公共业务，单点登录和企业支付网关等公共业务也属于一条产品线。

（4）六个集群：Web 集群、SOA 集群、中间件集群、数据库集群、Job 集群和 ITD 集群。

以上横向和纵向的划分形成了一个矩阵结构，也基本确定了基础架构和网络设计，接下来的四点是实施注意事项。

（1）核心自建，与应用核心价值相关的基础设施且需要定制的一定要自建，如外网负载均衡、数据库。

（2）立项，每一个大的阶段都需要立项，当作全新的项目实施，而不仅仅是上云项目的一部分。

（3）应用架构不变，核心网络架构重塑。尽量不要因为迁云而改变应用架构，哪怕只是单点问题，我们不可能因此而大量地调整代码，建议上云后再单独立项优化。

（4）早期适当浪费，后续缩减。迁云的过程可能会爆发一系列不可知的问题，为了更顺利地完成迁移，更快地定位问题，早期可以适当地浪费一些，待迁移稳定后再降级。

## 20.5 成功上云

我们将具体的迁移工作当作项目分七步来实施，包括规划、发布、测试、冻结、迁移割接、监控和结项。每一步都明确相关部门和责任人、具体工作内容、开始和完成时间。整个过程从 9 月 5 日到 10 月 21 日，历时一个多月，通过大家的共同努力和密切配合，最终成功将业务迁移到云端。具体方案与实施如下图所示。

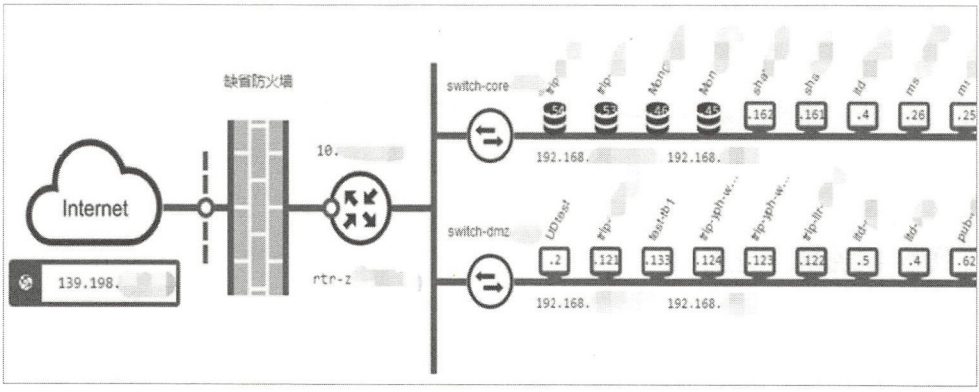


非卖品，仅供非商业用途或交流学习使用



| 序号 | 事项 | 描述 | 相关部门 | 所需时间 | 完成时间 | 备注 |
|---|---|---|---|---|---|---|
| 1 | 规划 | 总体设计、清单、注意事项、动员启动会 | 运维部 | 10个工作日 | 9月5日 | |
| 2 | 发布 | 完成所有发布；主机网络环境准备，应用迁移，数据库迁移 | 运维部 | 3工作日 | 9月8日 | |
| 3 | 测试 | 完成第一次测试，运维、开发、质量测试 | 运维、开发、质量部 | 10工作日 | 9月23日 | |
| 4 | 冻结 | 26号发布冻结，27-30号前完成第二次测试：高质量测试、产品业务测试；完成迁移前准备。 | 开发、质量部 | 5工作日 | 9月30日 | |
| 5 | 迁移 / 割接 | 9号准备迁移，12号迁移完成 | 运维、开发、质量部 | 4工作日 | 10月12日 | 如果失败，执行回退方案 |
| 6 | 监控 | 进行迁移后的业务监控，发现问题及时调整 | 运维、开发部 | 6工作日 | 10月20日 | |
| 7 | 结项 | 结项会议，输出总结报告 | 运维、开发、质量部 | 1工作日 | 10月21日 | |

整个过程中没有出现大故障，做到了用户无感知。同时解决了上云前的诸多问题，包括资源问题、平台问题和网络问题。故障明显减少，提高了产品的可用性。最终部署截图如下。




## 20.6 上云总结

以上过程，简单地总结如下：

- 内部虚拟化，外部云化。明确一个目标，然后找个靠谱的人来执行。
- 选择适合自己的云厂商。我们当时选择的是 xx 云，一对一 B2B 式的服务，混

非卖品，仅供非商业用途或交流学习使用



合云方面技术也不错。

- 20 倍规划、5 倍设计和 1.5 倍实施。确定一个指导性原则，比如上云八条。不仅可以指导实施，解决思想冲突，而且还会有一个很好的历史传承性。即便运维人员离职流动，也不会较大影响平台整体架构。
- 充分运用 PaaS 拉近与大企业的技术差距。中小企业的人员配置有限，我们应该更专注于自身的核心业务，多借力于云平台的基础服务。
- 核心自建，避免关键技术依赖。哪些要自己做，哪些使用云提供商的服务，需要进一步明确。与应用核心价值相关的技术一定要自建，不要偷懒。例如我们自建的有应用监控 Metrics、调度 Job、消息队列 RabbitMQ、微服务 MSA 等，下图是具体清单。

| Item | 具体技术 | 谁来做 |
| --- | --- | --- |
| 应用监控 | Metrics + InfluxDB + Grafana | 自己、APM（听云） |
| SLB、ILB | out-HAProxy、in-HAProxy | 自己 |
| 微服务 | MSA+API Gateway | 自己 |
| NoSQL | MongoDB | 云提供商 |
| 缓存 | Redis、Memcached | 云提供商 |
| 消息队列 | RabbitMQ | 自己 |
| 调度 | Job | 自己 |
| 数据库 | MySQL | 云提供商 |
| 数据库2 | MSSQL | 自己、历史 |
| 集中式日志 | ELK | 自己 |
| 服务器监控 | Zabbix | 自己、应用级别 |



非卖品，仅供非商业用途或交流学习使用

# 21 技术与业务的匹配与融合

## 21.1 技术人员与业务人员的抱怨

是什么在驱动公司的发展？带着这个问题，笔者跑去问技术人员，技术说“这是互联网时代，当然是技术，科学技术是第一生产力嘛”。笔者跑去问市场人员，市场人员说“没有市场，哪来的业务，要技术有何用”。笔者又跑去问运营人员，运营说是他们自己。最后碰巧遇到 HR，HR 说是“人”在驱动公司的发展。应该说他们说的都是正确的，但不够全面。这如同盲人摸象一样，引发了不少的争论，也直接或间接地导致了技术与业务的矛盾。

技术人员对业务人员的抱怨常有：

- 需求不明确，都搞不清楚他们想要什么；
- 业务做不好怪技术人员，技术人员是背黑锅的；
- 什么功能都想要，没有优先级；
- 我们加班加点开发出来了，最后他们不用。

同样，业务人员对技术人员的抱怨常有：

- 我们明天就想要，技术人员却说要开发 2 个月；

非卖品，仅供非商业用途或交流学习使用

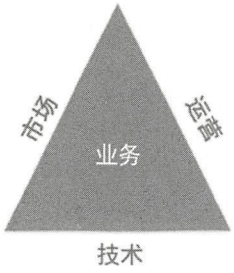


- 问题这么多，系统真垃圾；
- 功能这么少，别人有的我们没有；
- 速度那么慢，竞争对手那么快。

相信这些抱怨不仅存在于我所在的公司，同样存在于你的公司。而其他的跨部门沟通如人事、财务、法务，或多或少都会存在诸如此类的问题。

## 21.2 问题出在哪里

业务是什么？运营不是业务，市场不是业务，技术也不是业务，那业务到底是什么？维基百科的解释是“企业运用科学方法和生产工艺生产出可交付用户使用的产品与服务，并以此为企业带来利益的行为”。所以，业务应该是属于公司的，而不是属于某个部门的，运营和市场是业务方，而不是业务。我们在一起，是为了一个共同的理想、共同的业务目标，我们叫同事。运营、市场、技术三者互相依赖，缺一不可，不可分割，如果任何一个环节出现问题，就会影响整体业务目标的实现，也会引起互相的抱怨。运营、市场、技术三者的关系如下图所示。




每一个人眼中的对方是不一样的，在市场人员眼里技术是网吧，在产品人员眼里技术是工人，但在技术人员眼里自己是科研工作者。换个角度看，在市场眼里自己是超人，在产品人员眼里市场会“吹水”，在技术人员眼里却是“美腿”。可是，他们在竞争对手的眼中，也许都是“××”。每一个人都有自己的优点，研发人员逻辑性强，市场人员灵活、懂公关，运营人员则更加接近用户，做事也比较细。我们都需要先放下傲慢与偏见，然后增加彼此的了解和沟通才能更好地分工合作，分工就是每一个人做好自己的本职工作，跨进半步就是合作。

非卖品，仅供非商业用途或交流学习使用



## 21.3 理解源于彼此的了解

理解源于彼此的了解，我们要遵循事物的发展规律，要了解业务的发展规律，要了解技术的发展规律，以及彼此对对方的要求。

以下是业务自身的发展规律，分为 4 个阶段。

（1）起步阶段：这个阶段要快速验证商业逻辑、商业 idea。

（2）发展阶段：这个阶段需要快速地开拓市场，需要资本的注入，需要大量的钱来做推广。

（3）成熟阶段：精细化运营，提高效率，节约成本，还有风险控制。

（4）衰退阶段：需要转型，需要创新，需要快速地退出。

以下是技术的特征与发展规律。

（1）提前布局：“业务说明天就想要，研发说要开发 2 个月”。

（2）反馈修正：“刚上线就有这么多问题”，刚上线才那么多问题。

（3）系统观：“我的建议很好却没被采纳”，多方调研与评估。

（4）批量化：万一呢？技术擅长于批量化，“万一”的研发成本很高。

以业界为例，微软的产品一般到了 V3.0 才能用，1.0 版、2.0 版都不好用，但微软还是要发布。iPhone 在 2010 年之前不好用，Android 在 2012 年时也不好用，但是它们现在都很好用。如果业务人员不认识到这一点，不了解技术的特征，不了解技术的工作流程，则自然会对技术人员有误解，说不配合他们的工作。

业务规律对技术的要求：在起步阶段要新、要快，只开发核心功能 MVP，然后快速迭代，快速验证商业逻辑；在发展阶段要有营销功能、粗放式运营功能，一定要有订单量，做技术才有价值；在成熟阶段要精细化运营，要有大数据和分析功能，以及自动化功能。有了数据然后才需要分析，以自动化来节约成本和提高效率；在衰退阶段要转型，需要功能创新，减少系统维护。以上总结为八个字：销量、服务、管控、沉淀。即先以销量功能为主，然后做服务功能，接下来是管控功能，最后沉淀为基础业务服务，比如



非卖品，仅供非商业用途或交流学习使用



以“机+X”的形式存在，机票作为基础业务，X 作为创新点。

扩展开来，业务规律对其他方面及部门的要求：在人才方面，早期需要创业开拓型人才，中后期需要管理运营型人才，不同的阶段、不同的情景需要不同的人才；在财务方面，早期宽、灵活，后期要标准化、管控，如果早期就有各种复杂的报销流程，则会阻碍公司的快速发展；在法务方面也是如此；在管理方面，创业阶段要多折腾，要多破多立，当发展到一定阶段乱了才需要管理，如果一开始就管得过于严厉，则可能把它管死了；在流程方面，创业是找路，流程是修路，现在有不少组织提倡去 KPI、去流程、去级别，原因就是要创造，先找到路，然后才修路。

有时候不是技术人员不响应，也不是业务人员提的功能需求不对，而是所处阶段不同。不能要求一个成年人做小孩子的事情，也不能要求一个小孩子做成年人的事情。一个阶段要做一个阶段的事情，不清晰或产生矛盾，可能是因为没有定好位。

## 21.4 如何去匹配与融合

仅仅了解彼此是不够的，技术与业务需要匹配和融合。那什么是匹配？又如何匹配？匹配是阶段性的匹配，以时间为轴。先有商业逻辑，然后有产品逻辑，最后才有程序逻辑。如果没有商业逻辑，或没有表达好，则研发人员自然是不知道如何下手的。技术与业务的步调要一致，把大公司的系统拿过来直接就能用吗？系统是大脑，市场是手，运营是脚，如果只有灵魂，没有手和脚，则业务也应该很难运作起来。我们需要先确定一个阶段性业务目标，然后提前布局进行技术研发，最后快速反馈和改进产品，这样才是正确的路径。

先匹配再融合，融合是阶段内的融合，先把大方向确定了，然后在阶段内进行细节的融合。技术与业务该如何去融合呢？方法很多，以下是我们尝试过的 4 种方法。第一种是互训，各部门互训，你了解我的工作流程，我了解你的工作流程，互相了解各部门的输入与输出；第二种是轮岗，去别的部门轮岗、执岗，做其他部门做的事，做几遍才能真正地了解；第三种是活动，如打羽毛球、户外爬山，业务人员与技术人员交朋友，鼓励多一些非正式的沟通；第四种是思想突破和自我要求，这是最重要的一点，是思想的提升和自律。



非卖品，仅供非商业用途或交流学习使用



思想突破，改变传统观念，我们借助 5 个问题来引发不同的思想。

- 第一个是业务人员要懂技术吗？财务懂金蝶软件，人事懂打卡系统，研发人员又不懂这两个系统，所以业务人员也需要懂技术。
- 第二个是研发与技术的区别是什么？技术是第一生产力，研发中心是技术的重要部分，但不是全部。在互联网时代，技术如同电，这个能用那个也能用，每个人都需要了解技术。
- 第三个是研发部门是职能部门吗？什么是职能部门？财务和人事在电子商务公司是职能部门，但在财务公司和猎头公司呢？IT 部门在 IT 公司是业务部门，但在传统公司一定是职能部门，那在电子商务公司呢？
- 第四个是业务和技术的矛盾能否彻底解决？良性的矛盾、不同的视角能够促进公司的发展。
- 第五个是领导说各部门要多配合，那多配合就好吗？配合=匹配+融合，配合有配合不足、配合过度、配合恰当之分。过度配合会导致职责不清，交互频率高，效率变低，会产生“三个和尚没水喝”的情况；配合不足会产生缝隙；恰当的配合则是各个部门做好自己的事，再向前半步。

对他人的抱怨转化为对自己的反思，对他人的要求转化为对自己的要求，业务部门和技术部门都应该有一些自我要求。

**业务人员应有的自我要求：**

（1）在电子商务公司，业务人员也要懂技术，但可以不会写代码。业务经理也需要了解研发的基本流程，梳理并表达好自己的业务逻辑。

（2）懂得如何向研发中心提需求和反馈问题，懂得如何借助技术来提高市场的推广效果和运营管理水平。

（3）业务经理及其以上级别人员应该每月到研发中心走动一次，至少每季度需要走动一次。

**研发中心应有的自我要求：**

（1）树立只有业务发展了，研发才能发展的理念。“淘宝”为什么做得好，是因为



非卖品，仅供非商业用途或交流学习使用



它们有那么大的体量和业务场景。

（2）以服务为导向，强调技术价值的输出，产品没有人用就没有用，因为不产生价值。研发人员应该主动了解公司业务，学习公司业务。鼓励跨部门组织活动，比如羽毛球活动、户外爬山活动，这些非正式的沟通有利于彼此了解。

（3）产品经理是“媒婆”，在研发部门要说业务部门的好、业务部门的难，在业务部门要说研发部门的好、研发部门的苦，只有当好了这个“媒婆”，才能做一个合格的产品经理。产品经理每周要到业务部门至少 3 次，待 4 个小时以上，到业务部门执岗、轮岗，跑市场，搞运营。

没有配套的市场推广和运营计划，产品基本“没戏”。没有技术支持的市场推广和运营管理，那是劳动力，不是生产力。所以，只有技术和业务的匹配与融合，公司才有大的发展，科技才能转化为生产力。

## 21.5 什么在驱动公司的发展

谈了这么多，我们再次回到最初的问题，到底是什么在驱动公司的发展？3 种不同的看法，3 种不同的境界。第 1 种是“老子天下第一”，我来驱动，这是我最怕的一种，很难沟通，无知者无畏。第 2 种是不同阶段有不同做法，早期产品，中期市场，后期运营。第 3 种是投资人的要求，是竞争对手，是行业，是社会需求。竞争对手是友商，竞争对手与你一起前行，逼着你改进。社会需求是企业存在的根本，创业就是要解决别人的一个问题，社会共同的问题。另外，关于市场、运营、技术哪一个驱动好的问题，这个要看行业，要看业务自身的规律，老板的出身和喜好，各方力量的博弈。每一种做法都有很多成功的案例，以 BAT 为例，百度应该算是技术驱动型，阿里是运营驱动型，腾讯是产品驱动型，它们都有各自成功的理由。没有标准答案，存在即合理，有利于公司和社会的发展就好。



非卖品，仅供非商业用途或交流学习使用

# 22 研发团队文化是怎么“长”出来的

从死气沉沉到充满激情活力，从故步自封到好学分享，这是一个有关团队文化的主题。寺庙文化传承千百年，舌尖上的美食流传至今，它们是如何形成和生长的？是参考大公司或从管理书籍上挑选几个词语，还是脚踏实地，自己一步一步埋头干？本章与你一起探讨！

## 22.1 神秘的文化

平时经常听到 Google 公司讲企业文化，阿里、华为讲价值观，几个字就成为公司的核心竞争力，我们能不能自己也搞一个呢？其实文化只是一种思想观念、行为习惯、处事风格，家有家风，国有国法，一个民族有一个民族的文化。本章我们一起来探讨研发团队文化是怎么形成的，下面以笔者的经历来抛砖引玉。

笔者 35 岁前主要钻研技术，近几年逐步转向管理，带过几十人至一两百人的研发队伍。在最近的管理工作中，总结并归纳了几个可以贴到墙上的大字，即“共治分享自视一起拼，简单有效快”，接下来讲讲这 14 个字的来源。

非卖品，仅供非商业用途或交流学习使用



## 22.2 遇到的问题

笔者在一年多以前以技术负责人的身份入职了这家单位，到岗后发现了不少问题。

（1）技术差：架构臃肿，分层混乱，不利于维护和扩展；机房建设条件落后，有两个自建机房，没有故障处理流程。

（2）氛围差：团队没有活力，没有争论，没有学习和分享的氛围。准时上下班，人员懒散，没有激情，像一潭死水一样。不接受新事物，封闭自己，遇到问题相互推诿。

（3）组织结构不合理：人员没有充分利用起来，造成资源浪费，团队角色职责定义模糊，整个团队没有一个清晰的目标和落地的步骤，成员间相互的合作不协调。

## 22.3 解决之道

有问题当然要解决问题，所有的问题本质上都是人的问题，空降的领导必然会导致一部分人员的流失，只要将损失控制在一定的范围内即可。不是因为你不够好，只是因为他对过去领导的依恋，以及不正面看待新事物。改变一个人的成本，比招聘一个人的成本还要大。所以如果有人要离职，同意即可，让他优雅地离开，这是于公司于个人都有利的事情。解决了人的问题，接着解决技术问题，组建架构部，引进一些优秀的人才，自己带几个项目，打两场硬仗，培养兄弟感情的同时，也是让大家认同你的最好方式。做完了这些，然后调整组织架构，实行弹性工作制，接下来就是日常的管理工作了。部门共治，搭建统一工作平台，树立新工作作风，搞氛围，激活团队，只能一步一步地来。

### 1. 部门共治

部门共治就是部门要共同治理，笔者一直认为，每个人只要管好自己，部门的管理就简单得多。先在文件服务器上建一个文件夹，取名为部门共治，然后在里面新建如周报、周会纪要、工时统计、故障报告等目录。

（1）周报。有人可能会疑惑，研发还要写周报？其实，周报是一个很好的自我管理工具，上一周总结、下一周计划，让自己有一个清晰的工作目标。理清事情的轻重缓急，同时透明化自己的工作，增加彼此的信任。笔者一直有写周报的习惯，先是自己写，然



非卖品，仅供非商业用途或交流学习使用



后分级逐步推广到全员。

（2）工时统计。工时统计是每个月对团队工作时长的统计，由人事协助出报表。它统计到每个人、小组、部门的上班时长情况，主要通过疲劳指数、超出率、小组平均超出率、总平均超出率4个指标来体现。一切以数据说话，增加人员要看小组平均超出率，员工累不累看疲劳指数，总平均超出率体现了整个部门的负荷情况，了解整个部门的工作节奏，也为增员或裁员提供了参考指标，工时统计还可以与工单统计相结合。在具体操作时，需要注意可能产生的负面影响，比如造成团队成员的反感、过度鼓励加班等，其实超出率过高或过低都不是很好的现象。

（3）周会。管理小组每周碰一下，汇总一下工作和问题，同时提交会议纪要，全员都可以查看，并邮件方式同步给领导。我们的周会在每周一下午上班的第一时间召开，并要求开启午休关掉的灯，管理层从自己做起，严格约束自己，给大家树立一个榜样，顺带还解决午休延迟开灯的问题。早期的周会由我主持，后面轮流主持和记录。主持人站在部门负责人的角度，对汇报人员提出质疑或建议，这样就迫使主持人站在整个研发中心的角度去思考，培养了每个人的全局观，减少本位主义和平时工作中不必要的冲突，同时提高了管理人员的组织协调能力。

（4）故障分析报告。将故障分级管理，然后尽量透明化，重要故障全员通告，此工作由运维部门负责。故障报告不仅可追溯，也是一个总结分析的过程，可协助查找到根本问题。透明化可以提高大家的警惕性，造成的损失也间接地表明了代码与商业价值的关系。

（5）书籍分享和书斋建设。如果要构建一个学习型团队，则书籍的分享和书斋的建设是很好的办法。同时它相当于一面文化宣传墙，利于人员的招聘。书籍可以公司出钱购买，也可以鼓励大家捐赠、分享。书籍本身很便宜，但真的要捐赠时也有不舍，所以鼓励大家离开时可赎回。捐赠和购书，以及分享书籍的过程，也是学习氛围养成的过程。

### 2. 搭平台，立作风

搭工作平台，立新工作作风。没有度量就没有提升，管理要数据化、工具化。搭建统一的工作平台，提高大家的工作效率，同时要树立新的工作作风，比如站立式会议、周五分享日、项目管理工具和知识管理工具。



非卖品，仅供非商业用途或交流学习使用



（1）项目管理工具 JIRA。引入项目管理工具 JIRA，可以管理需求、工单、缺陷等各种任务。可以实时地监控、跟踪任务的状态，还可以在每月底出工单统计报告，评估开发人员的工作量和测试人员的工作量。

（2）知识管理工具 Wiki。类似于维基百科，主要用于知识的分享。可以创建部门的空间、自己个人的空间，写自己的工作心得、新的技术知识等。笔者的一些心得和重要邮件也会截图分享到自己的个人空间，这有利于新成员的了解和合作，也利于团队文化的形成。以前企业文化建设主要是搞海报、杂志等方式，现在采用这种方式更容易让人接受，也更容易推广。

（3）站立式会议。研发是一个需要创意且思维密集性的工作，所以技术讨论是必不可少的部分，而站立式会议是非常好的形式。小组周会或几个人讨论一个问题，非常适合站立式短会。20 分钟左右，大家站在白板前，不需要做会议记录，直接用手机拍照，会后发邮件或 QQ 群。简单高效快捷，大家保持专注，也不会懒洋洋的样子，没人会玩手机。因为站着，所以会议不会太长，高效、方便、随时随地，不一定需要大的会议室，所以研发中心要多一些白板，多一些小的会议室。

（4）周五分享日。技术的分享现在已成为研发团队的标配，每周五是我们技术分享日，时间定在 17 点到 19 点，这样避免影响正常的工作。同时让大家知晓，成长不仅是团队的事情，也是个人的事情。分享有利于主讲人的知识的巩固、总结，以及成果的展示，也有利于大家快速地学习。架构部的分享是工作的一部分，其他的则是自愿报名，自由组织。我们的分享也要采用简单有效快捷的方式，鼓励直接“show”代码，或者使用 Wiki 来讲，这样也利于二次传播，不一定要“高大上”，也不一定要 PPT，达到目的和起到效果即可。

### 3. 搞氛围，激活团队

工作和生活要平衡，认真工作，快乐生活。如果管理是鞭策或大棒，那么领导就是鼓励或胡萝卜。季度会议、乐捐与周四下午茶、户外兴趣小组是搞氛围、激活团队的很好的方式。

（1）季度会议。周会说事，季度会议谈情怀。每 3 个月左右开一次季度会，方式是新人介绍、上季度的总结、下季度的计划，其他如新书推荐、特长展示等。新人可以快



非卖品，仅供非商业用途或交流学习使用



速地融入整个部门，感受团队的热情。上季度的工作总结和下季度的计划是每个部门领导向下级汇报，我们鼓励上级向下级定期汇报。让每个人都知道各小组最近在干些什么事，下阶段要做什么，设置一个共同的目标，然后一起拼搏。同时主持人会向大家介绍近期自己读过的书，以及读书的感悟与心得，一起成长。

（2）季度会议如同年度会议，年会不开会，只是吃吃喝喝，以及节目表演，所以季度会议的形式也要轻松一些，以乐观向上的氛围为主，参与的人数比较多。我们的季度会议采用轮值制，每次的风格都不一样，也非常锻炼主持人的组织协调能力，会议一般占用周五分享日的时间。

（3）乐捐与周四下午茶。乐捐是一个自罚约定、问责制度，让员工敢于担当。对于开会迟到、系统小故障，乐捐能够起到很好的作用。惩罚只是手段，改进才是目的，培养承担责任的习惯，每次出现问题都有人站出来，故障也更容易明确主要责任人。乐捐不可强制，比如会议迟到你不愿意承担，那便需要由会议主持人承担。这是一种社区式的潜在力量，可借助红包等简单快乐的非正式方式。乐捐的钱可汇入部门基金，用于部门的下午茶。我们每周举办一次下午茶，周四进行，因为周一是例会日、周二是发布日、周三无会议日、周五是分享会。下午茶给大家营造一个轻松的氛围，促进交流，费用由乐捐、项目经费等共同组成。

（4）户外活动兴趣小组能够促进大家的交流，非正式的沟通能改善大家的关系，增强团队的凝聚力。我们组建了羽毛球兴趣小组、户外徒步兴趣小组，以及参与公司的年会表演，组织形式是开放的。兴趣小组有利于业务与技术的融合、跨部门沟通，大家劳逸结合，一起爬山一起拼。

#### 4. 更多管理工具

我们每一个管理工具针对的都是当前的问题，推行 2 至 4 周，一个阶段仅推广一个管理工具。我们尝试并留下来的管理工具总共有 20 多个，具体有：

周会周报、弹性工作制、技术分享会、季度会议、下午茶、乐捐约定、白板前站立式会议、技术评审制度、会议轮流主持制度、书斋建设、项目管理制度的推行、项目奖金申报制度、招聘比自己更优秀人的约定、绩效考核制度、末位淘汰约定、鼓励争论、跨部门沟通约定、源代码统一管理、Y 盘文档管理、JIRA 和 Wiki 研发管理工具推广、羽



非卖品，仅供非商业用途或交流学习使用



毛球兴趣小组、户外爬山兴趣小组、个体绽放，周一例会日、周二发布日、周三无会议日、周四下午茶、周五分享日。

## 22.4 总结与提升

太阳底下没有新鲜事，这些工具或措施，大家或多或少都知道一些，我们只是把它落地贯彻，形成了一定的体系。我们的团队一步一步往前走，确实越来越好。这时不断有新人加入，如何将这过去一年多的想法和做法都留下来，并传达出去呢？我们需要把它浓缩成几个字才好，这就是前面提到的“共治分享自视一起拼，简单有效快”，具体内容如下。

- **共治：**共治就是部门要共同治理，自我管理和部门管理一起进行，每个人要管好自己，我们希望部门的共治能转化为个人的自治。具体形式有部门共治文件夹、轮值会议等。
- **分享：**因为专业所以自信，因为自信所以开放，因为开放所以分享。具体形式有周五分享日、书斋建设。
- **自视：**自视就是自我察觉。对公司和他人的要求转化为对自己的要求，少一些抱怨，多找一些方法，如乐捐。
- **一起拼：**一起拼就是团队要一起拼搏，一起奋斗，一起喝酒，一起爬山，如户外运动、项目管理。
- **简单有效快：**把复杂的事情变简单，追求事物的本质，这就是简单有效。“快”是指快速地交付，快速响应业务的需求，比如站立式会议等。

以上是我们近一年的总结和实践，也许它并不完美，但它还是能表达团队之前及现阶段的一些行为。当我们确定了这个阶段的部门文化后，团队氛围有了较大的改善，从死气沉沉到充满激情活力，从故步自封到好学分享，从互相推诿到勇于担当。在后期的几个大项目中，项目的成功显得自然得多，特别在新员工的融合上，成本也低得多。

## 22.5 “长”出来的团队文化

可能现在你已发现，原来文化的构建并没有想象的那么难，也没有多少“高大上”。



非卖品，仅供非商业用途或交流学习使用



它需要脚踏实地，一步一步埋头干。先有管理工具、制度和行为措施，然后予以贯彻，形成一种习惯，最后才是总结提升。我们不能简单地参考大公司的做法，或者从管理书籍上挑选几个词语，然后领导喊两声或挂在墙上就万事大吉。这个过程就如同花朵一般，需要播种、栽培，然后才有收获。这样“长”出来的文化，才能管人做事，深入骨髓、改变思想，才能成为公司或团队的核心竞争力。以上浅见，希望能给你一些参考。



非卖品，仅供非商业用途或交流学习使用

# 后记

对我而言，写书要比写代码难多了，加之文字功底非常有限，出书确实是一件不容易的事情。我今年 37 岁，这本书算是我人生上半场的工作总结，十二年我只在一个行业干一件事。

所谓实践出真知，本书的内容全部源自我一线的摸爬打滚，也许并不“漂亮”，但绝对真实，内容涉及几百个应用、100 多个库、200 多位研发人员，是我在最近两家公司的真实经历。从自主研发到基于开源，从不可复制到快速搭建，坚持代码比文章重要，简单实用比炫技重要，基于常用场景而不是特殊场景，追求一篇文章即可快速地入门。文章完全站在程序员学习和使用的角度，以及架构师价值输出的角度，尽量提供 Demo 和设计案例，并且全部放到 GitHub 上对读者开放，希望对公司创造正面的、可直接使用的价值。

当然，本书还有很多不尽人意的地方，比如内容由独立的篇章组成，有些背景介绍重复，缺少中间件设计思路、更多应用案例、更大规模的验证，以及进阶篇的绩效考核等。每次在推广中间件后，我都会随机讲讲故事：这个中间件的起源，为什么要使用它，没有它会导致什么，用它带来了什么好处，行业里哪些公司在使用它，等等。让读者不仅知道如何使用，还能知道为什么、如何去用好。这些零散的过程并没有在书中很好地展现。另外，应用案例偏少，技改之路篇侧重于过程，应用架构设计篇侧重于概要设计与领域模型，虽然有机票垂直搜索引擎和上云纪要，但总体还是偏少。如果有专门的业务应用篇，以业务问题为导向来介绍产品、订单、价格和库存等，讲透技术为业务服务的精髓，那将是非常棒的事情。再者，缺少更大规模的验证，我们当时日订单峰值突破 10 万张，资金流水



非卖品，仅供非商业用途或交流学习使用



每年 300 多亿元。更大规模的验证需要业务更大的发展，在我接触的系统中，日订单量突破 200 万张，除书中类似功能的中间件外，自主研发的数据库分库分表中间件（类似 Mycat）也起到了非常关键的作用。世界上没有完美的系统，书自然也是这样，问题点就是我的下一个进步点。

这本书得以出版，要感谢的人太多。首先要感谢 InfoQ 和聊聊架构的编辑，是他们给了我创作的动力。感谢我原来任职的两家公司，给我提供了这么好的平台和机会。另外还有以前的同事，包括许珍珠、娄振宇、仝杭周、杨丽、邹振锋等所有曾经的同事，我们在一起做了大量早期研究工作，这些代码和研究成果才是书稿的基石。特别是杨丽，如果没有她的帮助，我可能无法独立完成全书。当然还有出版社的陈晓猛编辑，有了他的支持和协助，本书才得以出版。最后，还要感谢我的爱人，她和儿子的陪伴带给我好多快乐，并在文字方面提出了很多点睛之笔。有了这么多的机缘，才有了本书现在呈现的内容，谢谢你们！

张辉清

2018 年 11 月 30 日晚



非卖品，仅供非商业用途或交流学习使用

# 架构师进阶之路

在互联网发展的近二十年中，互联网技术这项伟大的发明从构想开始迅速发展，不停地将现实世界纳入它的势力范围。在 2018 年这个时间节点上，可以说互联网已经渗入社会生活的方方面面，对人类的活动产生了重大的影响。从社会经济发展的角度看，互联网重新定义了商业的运作方式，超高速的信息流转速度带动了商品和资金的高速流动，使市场经济的整体效率变得非常高，商务活动的互联网化为社会和企业创造了巨大的价值。

作为互联网行业的技术从业人员，在这个过程中“痛并快乐着”。技术在企业的竞争力中变得越来越重要，企业对技术研发的投入大幅提升，技术人员也越来越受到重视。同时，电商网站成为企业销售商品和提供服务的主体之一，甚至是唯一主体，网站构建的好坏事关企业生死，技术人员面临的挑战也变得前所未有的困难和复杂。

举例来说，复杂的电商业务带来了软件复杂性的挑战，快速迭代的业务流程带来了可扩展性的挑战，庞大的用户数量带来了并发性的挑战，7×24 小时的运营要求带来了可用性和可靠性的挑战，日益猖獗的互联网黑客攻击带来了安全性的挑战，等等。技术人员在构建网站的同时需要考虑以上所有的问题，并给出可靠的解决方案。这就好像在百米高空上走钢丝，每一步都需要小心翼翼，一旦踏错一步就可能产生不可挽回的错误。在这样苛刻的要求下，软件构建过程中的分析和设计，从形式主义逐渐变得具有现实意义。一些较大的 IT 团队因为分工更细致，甚至出现了专职从事软件分析和设计的岗位，也就是架构师。同时，软件构建中架构先行的理念已经深入人心。

非卖品，仅供非商业用途或交流学习使用



分享我本人的一些经验。我在从开发工作向架构工作转型的过程中，曾经遇到相当多的困惑，从技术到管理到业务，不一而足。原有的知识储备不足以支撑架构工作，即使补充了，大量的知识点如何梳理和组织、知识体系如何搭建也成为难题。由于软件架构设计的经验不足，无法判断每一个决策带来的风险，迅速地补齐架构经验的短板是非常困难的。在架构设计的每个细节中，如何平衡众多涉众的利益？这些涉众包括用户、业务方、兄弟团队等。在系统实施过程中，要怎样才能保证技术方案能够在开发、测试、运维等众多实施方之间达成一致，保证实施过程中不出现偏差？非常幸运，在这期间我和作者曾经有过一段共事的经历，我们有过很多关于技术、架构、业务和管理方面的讨论，让我对以上的问题有了更多的思考，受益匪浅。

本书忠实地体现了可以落地的架构理念，为一线的软件设计人员和技术管理人员分享了中小企业架构落地的方法论，同时结合了真实的项目经历，给出了可以采用的技术和管理手段。不仅如此，本书对于企业架构落地时常见的困难点，也就是组织架构的变更和技术团队的工程师文化建设有非常具体的建议，能够在技术团队建设上启发管理者。总而言之，所有从事互联网软件构建工作的读者，一定能从书中借鉴来自网站架构一线人员的经验和思想，本书是不可多得的实用读物。

——中骏置业　资深架构师　邓超

非卖品，仅供非商业用途或交流学习使用

# 谈谈互联网公司的技术架构和管理

互联网公司的技术架构涉及商业模式、目标用户定位和产品运营等，而且和公司所处的发展阶段息息相关。公司所处的具体发展阶段不同，相应的工作重心也不同。在创业初期，讲究发展用户慢、用钱慢、产品迭代速度快。中小互联网公司处于初创和发展期，正是需要快速建立技术基础、快速进行产品迭代的阶段。这个阶段，有张辉清这样的过来人手把手教你建立框架、架构和技术管理体系，无疑可以大大加速技术的成熟，为业务发展提供坚实的技术保障。

互联网公司的技术架构体系可以说已逐步成熟，规律性越来越强，同时不断推陈出新，从物理硬件到前端展现，精彩纷呈、博大精深。互联网公司的技术架构有它的目标、指导思想、规范和体系：

- 技术架构的目标是更好地实现业务发展的短期和长期目标，实现利益相关者的利益平衡。
- 技术架构的指导思想是立足产业、公司、项目的业务和系统的实际情况，本着“业务架构决定技术架构，核心业务流程决定主要技术架构”的原则，根据分布式计算的基本原理和实践，选择合适的技术方案来实现目标，具体体现为应用架构规划、UML、四加一视图、概念模型架起需求和实现的桥梁、拆分和缓存、设计模式和设计原则等，也就是所谓 Domain First，Persistence Second，

非卖品，仅供非商业用途或交流学习使用



Application Third。总之，这是充分调研、整体架构、增量迭代的过程。

- 建立技术架构的规范是为了统一公司的技术体系，总结规律、立规矩，便于技术的治理，防止各行其是和重复犯错导致技术成本的飙升，包括需求规范、架构过程和评审规范、开发规范、测试规范、部署架构规范、持续集成与发布上线规范、监控告警规范、事故处理规范等。
- 技术框架的体系涵盖物理层基础设施到展现层：与IaaS相关的有IDC自建机房、服务器集群、负载均衡、虚拟机、Docker 等；与安全相关的有防火墙、入侵检测、备份容灾等；与持续发布相关的有 GitHub、Jenkins 等 DevOps 自动化运维工具等；基础支撑软件有 Linux、MySQL、Hadoop、HBase、Elasticsearch（本书是 Solr，读者可以自行选择）等；中间件有 Java 的 Spring 系列框架、Tomcat，更有分布式架构必需的微服务框架 Spring Cloud 和 Dubbo、消息队列 RabbitMQ 和 Kafka、分布式缓存 Redis、分库分表中间件 Mycat 和 Sharding-JDBC、API 网关 Zuul、配置中心、调度平台等，还有各种常用组件的框架封装；此外，有为整个网站保驾护航的日志中心、APM 链路跟踪系统、度量平台、业务监控分析告警平台等。

互联网公司的技术问题主要有技术架构和技术管理两方面。在技术架构工作中，更多的是技术的深度和广度，而在管理性工作中，更多的是对于复杂的人和事的协调能力。互联网公司的技术管理工作主要有团队管理、项目管理和研发过程管理。

- 团队管理讲究客观、公正为主，人情为辅；决策必须科学合理；绩效考评既要看结果，又不能寒了人心。要靠事业、纪律、待遇、人情凝聚人心。主要依靠尊重人、激发人的自觉性来推动工作和团队管理，必要时也要剔除“刺头”来防止团队人心涣散。团队成员要德才兼备。

- 项目管理要防止“滑西瓜皮”，必须要有合理的项目计划，不断提高项目管理的可控度和可预测性。项目目标应适当，留有余量。项目的关键路径要确定清楚，优先确保完成，防止出现瓶颈，包括技术瓶颈、资源瓶颈、外部依赖瓶颈等。项目的时间要采取加班、加人、优化路径方案等必要措施以确保里程碑事件的落实。特殊情况下要调整项目计划，并第一时间通知干系人。项目的质量必须确保不能出现重大生产事故，这是由软件过程决定的，必须要有很好

非卖品，仅供非商业用途或交流学习使用



的工程意识。

- 研发过程管理主要是软件研发项目的整个生命周期的工程过程管理，包括需求、架构、开发、测试、上线、监控告警、事故及时处理和复盘等，需要规范严谨，尤其是配置管理、持续测试和发布过程最好实现工具化和自动化。

总之，管理首先要明确目标和规划，为此需要做必要的调查研究，实行民主集中制，从而做出正确的决策。管理目标、规划确定之后，最重要的是识人、用人、育人。这里首先需要对岗位的职责非常明确，也就是明确需要什么样的人。其次需要用各种手段来考察候选人的素质、能力和意愿是否符合岗位的要求。对于已确认的人才必须尊重，用人才最需要的东西来满足他。最后要育人。人无完人，我们不仅要包容人才的缺点，还要授之以渔，及时帮助人才，指出他们的优点和不足，及时指导他们的思想和工作方法，使其不断成长，能够胜任他们的工作。另外，用人做事，不是放任自流，还必须制定管理标准和制度，建立合理的机制、流程和文化，监控大局和关键的细节，形成体系，从而贯彻、执行和校验结果。所谓“制度管人、流程管事、机制文化决定一切”，最终靠结果说话。

中小互联网公司的技术架构和技术管理有其独有的特点。因为公司规模小，业务不够成熟，而产品需要快速迭代，所以应尽可能利用业界成熟的云计算和开源软件来降低运维和技术成本。但是，无论如何，基本的领域划分、服务化是必需的，至少要为将来的重构打下良好基础而不用把整个系统推倒重来，否则就会增加巨大的业务风险和成本。对于中小互联网公司的技术管理，应该保持队伍和流程的短小精悍、敏捷响应。

本书作者张辉清先生长期从事各种规模的互联网公司的架构和技术管理工作，历经架构师、技术总监、CTO 等技术和管理岗位，对于互联网公司的功能架构、非功能架构和技术管理有着丰富的研究和经验。对于中小互联网公司技术架构和管理的道与术，本书做了非常系统的阐述，其中包含很多付出了巨大心血和代价的宝贵经验，对于广大中小型互联网公司有根本性的指导意义。

——前隆科技　架构框架总监　徐刚

非卖品，仅供非商业用途或交流学习使用

# 短评

我和辉清之前在携程有过一段时间的交集，当时他是商旅研发部的架构师，而我是框架研发部的架构师。辉清的职业经历比较丰富，既做过开发和架构，又做过技术总监和 CTO。丰富的职业经历既锻炼了他各方面的能力，又拓展了他的视野。

在我眼中，辉清一直活跃在一线互联网公司的前沿，是实战型、偏业务型的架构师。辉清乐于分享，也具有良好的梳理总结能力，能够将他对技术、架构和管理的思考定期梳理总结出来，分享在 InfoQ 和聊聊架构等技术媒体上。

本书内容丰富，涵盖业务分析、领域建模、分布式系统架构、中间件和工具、微服务架构、技术管理及文化建设等主题。本书是辉清近几年在一线互联网公司生产实践的基础上，加上自己的系统化和体系化思考之后，沉淀下来的干货。本书对于一线架构师深入理解互联网分布式系统的架构设计并指导生产实践具有非常大的参考价值。

——微服务技术专家　拍拍贷基础框架研发总监　杨波

架构的落地、固化和提升，需要借助组织架构与技术架构的对齐来完成。从生产力到生产关系，从架构师到技术管理，我们的关注点也会发生变化——从框架、架构、公共服务和性能调优，到商业价值、技术的创新、技术与业务的融合，等等。这是一个架构师的进阶之路，也是辉清的心路历程，值得各位读者参考！

——饿了么 CTO　张雪峰

非卖品，仅供非商业用途或交流学习使用



在我接触过的众多技术人员里，能长期坚持总结的人并不多，能将总结与实践结合，并系统性地推动公司技术进步的人更少，辉清这方面的能力令我印象深刻。正因如此，他成长的速度非常快，几年之内，从一个工程师成功转型为架构师，并胜任数百人规模的研发组织的 CTO。这本书是他长期实践的结晶，不难看出，他已经在企业架构到应用架构改造，再到各种中间件、框架、工具的运用，甚至公共业务设计，以及技术如何推动业务进步等方面，形成了自己一整套的方法论。对于大规模的研发组织，由于分工明细，并不缺少各个领域的专家，但着眼于全局的架构师并不多见。本书的完整性和体系性，非常适合中小型的研发组织借鉴，尤其是正在进行架构转型的传统企业的研发团队，一定能通过本书受益良多。

——平安好医生研发总监　刘剑

这本书从头到尾，从我这个老工程师的角度来说就是两个字——干货，非常适合处在成长期且比较迷茫的技术工程师。这本书的价值在于它透过一个老技术人的多年实操和沉淀，让你一下子就切入一个思路明晰的方法论。本书最后几章可是我老同事的进一步进阶了，“懂了”距离“消化”，还要通过实际操作的历练。俗话说得好：师父领进门，修行靠个人。希望大家能早日成为像作者一样的技术牛人！

——造艺科技 CEO　梁晓靖

代码混乱、结构不清晰、开发效率低、发布周期长、发布出错率高、排查问题困难等困扰着很多互联网研发团队，也曾是我和作者一起需要面对的问题。本书第 18 章技改之路，我是亲历者和见证者，整个过程我与作者一起拼搏奋斗，至今难忘，受益匪浅！

——洋码头资深架构师　戈建华

本书没有晦涩难懂的技术分析，而是通过一个个真实案例带你参与一次完整的技术改造，从研发团队的技术痛点着手，帮助中小型研发团队从无到有快速建立一套主流的技术架构。

——携程旅行网　仝杭周

非卖品，仅供非商业用途或交流学习使用



关于架构设计，百人有百人的想法，但我们还是希望能借鉴前人走过的路，从这么多人的想法中找到一些共性来更快速地确定自己的想法是否合理。这本书既有架构方法论，又有中间件研发和使用的实战经验。对于已经或即将在中小企业从事架构师职位的同行们，本书是一本非常实用的参考书籍，值得从头到尾认真阅读，并且相应地去实践，然后运用到实际的项目中。相信用不了多久，你也可以成为一名合格的架构师。另外，我有幸参与了本书第 5 章生产环境诊断工具 WinDdg 的早期研究，那是一段值得怀念的快速成长时光。最后，感谢张辉清先生能把这本书给写出来，非常不容易！

——同程艺龙　有票儿技术负责人　许珍珠

《小团队构建大网站》通篇以简单的实用主义“生吞”复杂业务场景，架构师肚子里的那点东西全被作者掏出。如果读者跟我一样喜欢在周末花上半天时间待在书店，挥一挥衣袖不带走一本书，那么你读完“开篇”之后悄悄放回书架就可以了。如果你刚入职一家不错的互联网企业，千万不要放过“架构篇”，尤其是“企业总体架构”一章，这会帮助你在一群绝顶聪明的“偏执狂”中找到自己的格调和定位。若能活用“框架篇”中所列举的开源组件，则大体上能够搭建一个不错的网站。但那远远不够，你至少要读懂这些组件的接口脉络和背后的设计思想，直至有一天你能够带领着三五个人将这些组件全部替换。“公共应用篇”中所列举的真实的案例，读者不必盘问出处，在互联网行业有些规则需要遵守和维护。“进阶篇”回归业务问题的本质，从代码的解耦到业务的解耦，再到团队建设的人文关怀，正是作者心路历程的真实写照。

——慧睿思通　资深架构师　张向明

本书面向有一定基础经验的开发者，介绍了构建大型网站涉及的方方面面。从顶层架构设计到单个项目架构设计再到应用分层规范，从中间件的应用到线上故障诊断，从技术改造到团队文化建设，可谓字字珠玑，娓娓道来，足见作者实战经验之丰富。书中某些章节偏向.NET 平台，但万变不离其宗，其他平台的开发者仍然能从中获取养分、吸取经验，是一本值得一读的好书。

——阿里巴巴　张智

非卖品，仅供非商业用途或交流学习使用



细细品完《小团队构建大网站》，作者结合了架构、业务和管理等众多维度，提供了一系列解决方案，自下而上，一步步“带领”中小团队快速构建高性能、高可用、可扩展的大网站。做事严谨、思路清晰、架构经验丰富是辉清的标签，这本书同样如此。作者根据自己十多年的架构和管理经验，真真切切地把握住了中小团队的命脉和痛点，非常优秀，相信读者看完这本书，一定会受益匪浅！

——同程艺龙　交通架构组负责人　曹爱虎

4 年前，在携程商旅事业部，业务系统需要进行架构重构，但留给技术部门的时间非常紧张（2 个月内必须完成），做技术的人都知道架构重构是一件非常复杂的事情，而且还要考虑兼容老系统。2 个月后，突然传来捷报说重构成功了，并且得到了业务部门的高度好评。系统不但稳定，而且可以支持 10 倍的业务增长，带领这支攻坚团队的架构师正是张辉清。我本以为这次重构是一个“卡位战”，以欠技术债的方式仅实现功能，但事实相反，此项目不仅具备高可用、高性能、高并发、高扩展等特点，而且有机地将技术和业务结合在一起，实现了技术驱动业务。本书是辉清近几年的经验总结，揭秘如何为中小公司赋予大型互联技术的能力。

——携程旅行网　数据智能部技术专家　陈昌

在这个技术浮躁的时代，众多的技术人员沉迷于一些奇巧淫技，忽略了架构的本质是合理地组织技术和人，更好地服务于业务。本书从实战出发，通过一个个实例阐明架构中的种种方法论如何落地，如何在架构落地的过程中保持技术的前瞻性及柔性，如何有效地避免过度设计。作者以 CTO 的视角，从业务和技术痛点入手，讲述了带领技术团队快速搭建小而美的整体架构的过程。本书背后的分析思想和设计思路，非常值得快速发展的中小团队借鉴。

——百度资深架构师　杜亚明

作为一个技术人，架构师是每个人追寻的目标，也是个人价值和成就的一个重要标志。那架构师是怎样炼成的呢？或者说，怎样才算是一个优秀的架构师呢？我们需要几年甚至十几年的一线技术工作经验，我们需要正好遇到公司重大技术重构的机会，我们

非卖品，仅供非商业用途或交流学习使用



需要正好遇到公司重大业务系统的升级计划，我们需要正好遇到……这么多苛刻条件的限制，造成架构师修炼的不易。那我们是否能找一本武功秘籍，让那些在技术路上苦苦追寻的技术人员少走弯路，甚至速成呢？从这个角度上看，《小团队构建大网站》无疑是一个非常不错的选择。它是辉清历年经验的总结，由浅入深，从思想到框架再到实施和管理，实用、实在，让我们能从简单的学习模仿，到融会贯通，然后进阶到自成体系。愿这些来自辉清的干货，可以为读者指出一条不同凡响的架构师进阶之路。

——驴妈妈架构 PU 总经理　高亚峰

架构的大道理听了很多，但依然不会操作。能不能将这些抽象的道理具体化、技术服务产品化，然后像使用商业产品一样使用它们呢？从“做”中学，先做再学，照着案例学，让各位同学快速入门。你可以不懂电视机的工作原理，但你依然会使用电视机。一本好书可以让人少走 3～5 年弯路，一本好书可以奠定一个人的大局观。站在前人的肩膀上，你可以看得更高、望得更远！

——申通快递总工程师　吉日嘎拉

随着互联网行业开源潮流的兴起，越来越多的公司通过技术大会、技术图书出版等方式分享架构思路和研发经验，几万人的超大型研发团队的技术架构思路，对于很多 200 人以下研发团队的公司来说值得借鉴，但由于研发资源与团队规模的差异，可能不容易直接应用。本书作者站在中型研发团队的视角，以 CTO 的身份亲历架构实践。这些架构思路与成熟开源组件可直接应用，相信能带来不错的参考价值。

——dnc 开源社区 CEO/CTO 联盟发起人　Mike

《小团队构建大网站》对于很多公司来说都值得参考和借鉴，大公司也有很多小团队。随着.NET Core 开源新时代的到来，越来越多的中小研发团队会使用 C#来构建业务系统。本书介绍了一套基于 C#、基于开源、可直接落地、可快速搭建的框架及架构方案，如果说大公司方案是定制的“劳斯莱斯”，那么这个方案就是经济实用的“大众”！

——微软 MVP，腾讯科技　张善友

非卖品，仅供非商业用途或交流学习使用



张辉清同学曾于某技术媒体发表过不少文章，心有戚戚焉则于江湖中得以认识，整体印象是做事行文都是绝对的实战派，偶尔也有俏皮幽默的案例出自其手。本书涵盖了架构方法体系、常用“防身”工具、项目案例、管理 tips 等。江湖上集群 TPS 极高的场景其实有限，按照二八原则，80%甚至 99%的场景都是中小型规模的应用。张兄视野横跨业务、技术、架构和组织，既有原则提炼，又有案例叙述，无空洞之感，无堆砌之累，是难得的于平实中见真诚的作品，尤其是把不少案例都开源出来了，实证精神可见一斑。祝各位读者有愉快的阅读旅程！

——蚂蚁金服高级技术专家　右军

本书内容相当丰富，不仅对想成为架构师的程序员有专业指导，也对已经是架构师并想转到技术管理岗的读者有参考意义。辉清同学对待技术踏实而低调，热爱经验总结与分享，是我们技术人学习榜样。

——《架构探险》作者　黄勇

互联网技术经过几十年的发展，已经从“铁器时代”进入“机器时代”。得益于开源运动的蓬勃发展，以及技术的日益开放，原本只有大公司才能拥有的技术和系统已经是“旧时王谢堂前燕，飞入寻常百姓家”了，中小团队甚至初创公司都能够基于这些技术和系统快速完成系统的开发，使团队能够更加聚焦于业务的发展。

但这并不意味着简单采用“拿来主义”就万事大吉，中小团队在构建系统架构的时候往往面临几个核心问题：首先，类似的技术和方案太多，具体该用哪个并不是一目了然的；其次，即使选定了具体技术或方案，如果没有经验积累，这些技术和方案的最佳实践和注意事项（俗称“坑”）是很难预先知道的；最后，构建一个完整的大网站需要的技术栈很多，如果没有系统的指导，则很可能是“摸着石头过河”，进入“踩坑—填坑”的循环。

本书是辉清多年技术、经验、思考和感悟的一个集大成的总结，涵盖了架构设计技术栈的方方面面，很好地解答了上述三个问题，具有非常强的指导意义，形象一点来说就是：照着做，你也能设计和 BAT 一样好的架构！

——《从零开始学架构》作者，资深技术专家　李运华

非卖品，仅供非商业用途或交流学习使用

## 好书力荐


**拒绝堆砌臃肿，支持纯正原创**

欢迎投稿：chenxm@phei.com.cn

非卖品，仅供非商业用途或交流学习使用

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

# 小团队构建大网站

## 中小研发团队架构实践

专家力荐

本书从实战出发，通过一个个实例阐明架构中的种种方法论如何落地，如何在架构落地的过程中保持技术的前瞻性和柔性，如何有效地避免过度设计。作者以CTO的视角讲述了带领技术团队，从业务和技术痛点入手，快速搭建小而美的整体架构的过程。本书背后的分析思想和设计思路，对于快速发展的中小团队是非常值得借鉴的经验。

——百度资深架构师 杜亚明

代码混乱、结构不清晰、开发效率低、发布周期长、发布出错率高、排查问题困难困扰着很多互联网研发团队，也曾是我和作者一起需要面对的问题。本书第18章技改之路，我是亲历者和见证者，整个过程我与作者一起拼搏奋斗，至今难忘，受益匪浅！

——洋码头资深架构师 戈建华

本书是作者多年技术、经验、思考和感悟的一个集大成的总结，涵盖了架构设计技术栈的方方面面，具有非常强的指导意义，形象一点来说就是：照着做，你也能设计和BAT一样好的架构！

——《从零开始学架构》作者，资深技术专家 李运华

本书从框架、架构、公共应用和性能调优，到康威定律、技术与业务的匹配与融合等，从生产力到生产关系，从架构师到技术管理，均有涉及，这是一个架构师的进阶之路，也是作者的心路历程，值得各位读者参考！

——饿了么CTO 张雪峰

本书内容丰富，涵盖业务分析、领域建模、分布式系统架构、中间件和工具、微服务架构、技术管理及文化建设等主题。本书是作者近几年在一线互联网公司生产实践的基础上，加上自己的系统化和体系化思考之后，沉淀下来的干货。对于一线架构师深入理解互联网分布式系统的架构设计并指导生产实践，本书具有非常大的参考价值。

——微服务技术专家，拍拍贷基础框架研发总监 杨波

博文视点Broadview
新浪微博 weibo.com
@博文视点Broadview

责任编辑：陈晓猛
封面设计：李 玲

上架建议：计算机/架构设计

ISBN 978-7-121-35215-7
9 787121 352157 >
定价：69.00元